DOCUMENT RESUME

ED 367 294 IR 016 554

AUTHOR Yoder, Sharon
TITLE "Linkway" Scripting: An Introduction.
INSTITUTION International Society for Technology in Education, Eugene, OR.
REPORT NO ISBN-1-56484-037-9
PUB DATE 93
NOTE 246p.; For related document, see IR 016 553.
AVAILABLE FROM International Society for Technology in Education, 1787 Agate St., Eugene, OR 97403-1923.
PUB TYPE Guides - General (050)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS Computer Assisted Instruction; Computer Graphics; *Computer Software; Educational Environment; Experiential Learning; *Hypermedia; *Programming
IDENTIFIERS *LinkWay

ABSTRACT

This book in intended for use by individuals who are comfortable with exploring new software and who want to learn LinkWay (one of a new generation of hypermedia programs) scripting. The book contains several chapters which are divided into three themes. The first part of the book is a review and starts with Chapter 0 providing a brief overview of LinkWay; the next two chapters provide simple material that is often covered in beginning books and workshops about LinkWay. The second theme includes chapters 3-20, and it centers on important but relatively easy-to-use ideas about LinkWay scripting and some of the tools that come with LinkWay. The final theme encompasses chapters 21-25 and focuses on the introduction of central ideas in programming and computer science: conditionals, looping, scope of variables, and parameter passing. A final chapter answers the question, "Where do I go from here?" A list of reference sources and an index to all important concepts and LinkWay commands or functions are included. Six appendices contain information on keyboard shortcuts, LinkWay commands, LinkWay functions, LinkWay operators, DOS extensions used by LinkWay, and using media with LinkWay. (TMK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
☐ This document has been reproduced as received from the person or organization originating it
☐ Minor changes have been made to improve reproduction quality
• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

ED 367 294

LinkWay Scripting An Introduction

120165S4

ISTE

BEST COPY AVAILABLE

Sharon Yoder

"PERMISSION TO REPRODUCE THIS MATERIAL IN OTHER THAN PAPER COPY HAS BEEN GRANTED BY
D. Moursund
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

About the Authors

Dr. Sharon Yoder has taught mathematics and computer science at the junior high and high school level for 15 years. Her most recent public school experience was as a secondary computer science teacher and computer coordinator involved in developing system-wide computer curriculum and in planning teacher inservice training.

In addition, she has taught mathematics, computer science, and computer education at a number of universities in northeastern Ohio, including Kent State University, the University of Akron, and Cleveland State University.

After a year as an education specialist for Logo Computer Systems, Inc., Sharon returned to teaching. She is currently at the University of Oregon, where she teaches computer education courses.

For the past several years, she has conducted workshops and presented papers at conferences nation-wide and has been involved in a number of book-publishing projects, including the *Nudges* series, *Introduction to Programming in Logo Using LogoWriter*, and *Introduction to Programming in Logo Using Logo PLUS*. In addition, she has been a frequent contributor to *The Computing Teacher*. She is currently editor of *Logo Exchange*.

Editing: Christy McMannis and Ron Renchler, The Electronic Page
Editorial Assistance: Anita Best
Book Cover: Tamara Kidd

© International Society for Technology in Education, 1993
1787 Agate Street, Eugene, Oregon 97403-1923
Order Desk: 800/336-5191
Phone: 503/346-4414
Fax: 503/346-5890
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu

ISBN 1-56484-037-9

PEST

LinkWay Scripting
An Introduction

Sharon Yoder

© International Society for Technology in Education, 1993

Contents

Introduction 1

- What Do You Need to Know? 2
- What Hardware and Software Do You Need? 2
- Who Should Use This Book? 3
- The Structure of This Book 4
- How to Use This Book 4

Chapter 0. A Brief Review 7

- Getting Started 7
- What Is *Linkway*? 7
- What You Need to Know to Begin 10
- A Brief Review—Creating a Folder 10
- A Brief Review—Creating a New Page 14
- A Brief Review—Using *Linkway Paint* 15
- A Brief Review—Editing a Base Page 19
- A Brief Review—Creating a Button 21
- A Brief Review—Creating a Field 24

Chapter 1. Buttons and Scripting 27

- Using Pop-Up Buttons 27
- Introduction to Scripting 29
- Debugging and Testing 33
- New *Linkway* Words in This Chapter 34
- Project Ideas 34

Chapter 2. Displaying Text and More 35

- The Prompt Command 35
- The Msg Command 36
- Functions 38
- Using the Text Command to Display Messages 40
- Debugging and Testing 42
- New *Linkway* Words in This Chapter 43
- Project Ideas 43

Chapter 3. Editing Text 45

- *Linkway* Text-Editing Keys 50
- Project Ideas 50

Chapter 4. Working With Objects 51

- Modifying Objects 52

Cutting, Pasting, and Removing Objects 54
Moving Objects 55
Debugging and Testing 57
New *Linkway* Words in This Chapter 58
Project Ideas 58

Chapter 5. Shortcuts and Tools 59

Keyboard Shortcuts 59
Using Expert Mode 62
Using Tools 63
Debugging and Testing 66
Key Shortcuts 66
Project Ideas 66

Chapter 6. Working With Fields 69

Selecting Objects 69
Putting Text in Fields 70
Working With Fonts 72
Debugging and Testing 73
New *Linkway* Words in This Chapter 74
Project Ideas 74

Chapter 7. Overview of *Linkway* Grammar 75

Commands and Functions 75
Syntax Descriptions 77
Objects 79
Object Order and Mouse Clicks 81
Scripts 81
Debugging and Testing 82
Project Ideas 83

Chapter 8. Document and Reference Buttons 85

Document Buttons 85
Reference Buttons 87
Using Triggers 89
Debugging and Testing 91
Editing Keys for LWEdit 92
Project Ideas 93

Chapter 9. Design Issues 95

Folder Structure 95
Base Pages 100
Conventions for Using Icons 101
Some Other Considerations 102

Debugging and Testing 103
Project Ideas 104

Chapter 10. More on Working With Objects 105

Object Properties 105
Coordinate Systems 108
Using Positions of Objects 108
The Mouse Position 110
Debugging and Testing 111
New *Linkway* Words in This Chapter 112
Project Ideas 112

Chapter 11. Writing and Using Procedures 113

Making *Linkway* Code Easier to Read 113
Using Procedures More Than Once 116
Testing Your Scripts 117
Debugging and Testing 118
New *Linkway* Words in This Chapter 118
Project Ideas 118

Chapter 12. Using LWCapture 119

LWCapture 119
Using Captured Images 121
Some Technical Matters 123
Project Ideas 123

Chapter 13. Working With Graphics From Scripts 125

Using Bar and Box 125
Using Line 127
Using Color Commands for the Background 128
Using Color Commands for Objects 129
Debugging and Testing 130
New *Linkway* Words in This Chapter 131
Project Ideas 131

Chapter 14. Printing 133

Printing a Page 133
Printing Using Tools 133
Printing Pictures 134
Printing Text Files 135
Printing From Scripts 136
Printing Scripts 137
New *Linkway* Words in This Chapter 137
Project Ideas 137

Chapter 15. Animation 139
Using Objects 139
Using Box, Bar, and Line 142
Flipping Pages 142
Debugging and Testing 144
New *Linkway* Words in This Chapter 144
Project Ideas 145

Chapter 16. Working With the Mouse 147
Modifying the Mouse Pointer 147
Mouse Buttons and Mouse Location 149
Debugging and Testing 150
New *Linkway* Words in This Chapter 150
Project Ideas 151

Chapter 17. Using Do and List Objects 153
Using the Do Command 153
Creating and Using List Objects 155
Creating and Using AutoExec Buttons 157
Debugging and Testing 158
New *Linkway* Words in This Chapter 159
Project Ideas 159

Chapter 18. Working With Pages 161
Linking Pages and Folders 162
Adding and Deleting Pages 163
Controlling the Screen 165
Debugging and Testing 166
New *Linkway* Words in This Chapter 167
Project Ideas 168

Chapter 19. More on Graphics 169
New Techniques in *Linkway Paint* 169
Picture Cut Buttons 173
Debugging and Testing 173
Keys Used in This Chapter 174
Project Ideas 174

Chapter 20. Interactive Programming 175
Using Input 175
Using Menu 177
Using Fmenu 179
Debugging and Testing 180
New *Linkway* Words in This Chapter 181
Project Ideas 181

Chapter 21. Conditionals 183

Single-Statement If 183
Multiple-Statement If 184
Single-Statement If-Else 185
Multiple-Statement If-Else 185
Using Random Numbers 186
Using the If Structure 187
Using Match 190
Debugging and Testing 191
New *Linkway* Words in This Chapter 192
Project Ideas 193

Chapter 22. *Linkway* Operators 195

Mathematical Operators 195
Logical Operators 196
String Operators 197
Operator Precedence 198
New *Linkway* Words in This Chapter 199
Project Ideas 199

Chapter 23. Variables 201

Variables 201
Using Variables 203
Blanks and Subscripts in Variables 205
Scope of Variables 206
Debugging and Testing 209
New *Linkway* Words in This Chapter 209
Project Ideas 210

Chapter 24. Looping 211

Looping a Specific Number of Times 211
Testing a Condition to Exit a Loop 213
Modularizing Your Scripts 215
New *Linkway* Words in This Chapter 216
Project Ideas 216

Chapter 25. Using Parameters 217

Getting Started With Parameters 217
Returning a Value From a Procedure 219
Debugging and Testing 221
Project Ideas 221

Chapter 26. Where Do I Go From Here? 223

Appendix 1. Keyboard Shortcuts 225
Editing Objects in *Linkway* 225
In *Linkway Paint* 225
In LWEdit 226
Appendix 2. Commands Used in This Book 227
Appendix 3. Functions Used in This Book 229
Appendix 4. Operators 231
Appendix 5. Extensions Used by *Linkway* 233
Appendix 6. Using Media With *Linkway* 235
Index 239

Introduction

The two software packages *LinkWay* and *LinkWay Live!* are tools for creating hypermedia documents on MS-DOS/PC-compatible computers. The more recent release of the software *LinkWay Live!* makes the integration of external devices, such as videodiscs, into hypermedia documents quite easy. Since the introduction of *HyperCard* on the Macintosh computer in 1987, there has been growing interest in packages such as *LinkWay*. Today, both teachers and students are using these new products in the classroom.

While products such as LinkWay, *HyperCard*, and their competitors are relatively new to educational computing, the underlying concepts behind these products are not new. All of these products allow the production of nonsequential documents. The idea of nonsequential text has been talked about for many years. The term initially used to describe such documents, *hypertext*, was coined by Ted Nelson in his book *Computer Lib/Dream Machines*, published in 1974.

Although most of the documents we read are sequential in nature, there are several types of hypertext-like books with which most people are familiar. One such example is a dictionary. Dictionaries are not read from front to back but rather are used in a nonsequential manner. Another familiar example of a noncomputer hypertext document is a "choose your own adventure" book in which the reader chooses the path through the story.

With *LinkWay*, an MS-DOS computer can be used to produce your own hypertext documents. In addition, it is easy to add graphics and sound to documents or to access videodiscs and other external media. Adding other media to nonsequential text documents produces documents that are generally referred to as *hypermedia documents*. *LinkWay* and *LinkWay Live!* make the creation of such documents quite easy.

LinkWay uses a mouse and pull-down menus. There is no need for users to be sophisticated MS-DOS users. Most of the features of *LinkWay* can be accessed by "pointing and clicking" with a mouse. In addition to its many menu-based features, *LinkWay* has a built-in programming language that allows users to create *LinkWay* scripts. This programming language is designed specifically for working in the *LinkWay* environment.

LinkWay Scripting—An Introduction is designed to introduce learners to the tools associated with *LinkWay* and to the writing of programs using the *LinkWay* scripting language. This book goes into much more detail than the *Quick Start* booklet that comes with *LinkWay*. It also goes into more detail than the *LinkWay* tutorials. However, it is not as detailed as the technical reference manual that accompanies the program. *LinkWay*

Scripting—An Introduction does not provide a mouse-click-by-mouse-click explanation for every example. Nor does it attempt to include everything that can be done with *LinkWay*. Rather, this book uses some simple examples to demonstrate the new material presented in each chapter. It provides suggestions of activities to practice new ideas. It connects *LinkWay* scripting to computer science concepts and other programming languages.

What Do You Need to Know?

Although the book begins with a brief review of *LinkWay* objects, it is *not* a beginning book. To use this book you should:

- be familiar with using a mouse on MS-DOS computers
- have worked through the *LinkWay* tutorial
- have used *LinkWay* at the "authoring" user level; that is, you should have explored some *LinkWay* folders and created some pages, buttons, pictures, and fields
- have begun to want to learn about *LinkWay* scripting in order to use *LinkWay* more extensively than is possible using the menu bar.

If you have not worked with *LinkWay*, you should spend some time with a beginning book, such as *LinkWay for Educators—An Introduction* (available from the International Society for Technology in Education).

Readers who have both extensive experience with computers and know one or more programming languages may find that *LinkWay Scripting—An Introduction* alone will meet their needs. However, even such experienced computer users should have a reference available to provide the background not discussed in detail in this book. The *Reference Guide* that comes with *LinkWay* will provide the necessary level of technical detail.

What Hardware and Software Do You Need?

The beginning of your *LinkWay* manual tells you what hardware configuration and what version of DOS you need to run your version of *LinkWay*.

> LinkWay *will run under* Windows, *but there is not a separate* Windows *version of the program.*

In addition to the required hardware and DOS version, you will need a mouse and the associated mouse driver. While it is possible to use older versions of *LinkWay* with floppy-disk drives, you can quickly find yourself doing a great deal of "disk swapping"—constantly switching disks so that

LinkWay can access the information it needs. Thus, it is best if you have a hard drive for your work with *LinkWay*.

Versions of *LinkWay* fall into two categories—older versions simply called *LinkWay* followed by a version number, and the newer product-release referred to as *LinkWay Live!* You do not need to have *LinkWay Live!* to use this book. Nearly all the activities covered in this book will work exactly as described if you use *LinkWay* 2.01.

> *If you are using a version of* LinkWay *with a number lower than 2.01, you may find some features that differ.*

Features unique to *LinkWay Live!* will always be noted.

Throughout the book the term *LinkWay* will be used to refer to both *LinkWay Live!* and *LinkWay*. The term *scripting* will be used to refer to writing programs in the *LinkWay* and *LinkWay Live!* scripting language.

The pictures of screens in this book represent screens in *LinkWay Live!* in a monochrome mode. If you are using an earlier version of *LinkWay* or if you are using a different graphics mode, your screen will occasionally look somewhat different. In general, the different appearance of the screen will not affect the way examples work.

Who Should Use This Book?

This book can be used by individuals who are comfortable with exploring new software and who want to learn *LinkWay* scripting for their own purposes. It is designed for self-study, preferably where the learner has other references, whether printed, online, or human. It can also be used in a classroom setting, such as an inservice training course, where an instructor is available to demonstrate techniques covered in the book and provide personal support as students learn. The book can also be used in secondary school classes with students who want to learn to program using *LinkWay* scripting, or it can be used for staff development classes.

As hypermedia environments become more and more common in our schools and other learning environments, increasing numbers of people will want to learn to use *LinkWay* to produce their own products. Increasing numbers of educators will want to know enough about *LinkWay* to help students produce hypermedia projects. Increasing numbers of adults will want to learn about *LinkWay* to create products that will aid them or their peers in their work environment. This book can be used to help learners of all ages develop more advanced skills with *LinkWay*.

Although in recent years there has been a decrease in the teaching of traditional programming in schools, *LinkWay* is increasingly being taught at a variety of grade levels. Learning *LinkWay* scripting can be a natural extension to using *LinkWay* menus to create projects. By learning *LinkWay*

scripting, students can learn some of the fundamentals of programming. It seems likely that many of the applications both students and adults will use in the future will include programming languages. Even now we have spreadsheet languages, database languages, and the ability to write macros for telecommunications programs. Increasing numbers of applications include a "scripting language" that allows users to personalize their software. The sophisticated computer users of the future will most likely be more productive if they learn some elements of computer programming. Thus, this book could be used in a variety of settings to help learners understand how to use *LinkWay* scripting with their *LinkWay* projects.

The Structure of This Book

Although the chapters in this book are numbered sequentially, the book really consists of three parts. The first part of the book is a review. Chapter 0—so named because the assumption is that the reader already knows what is covered in the chapter—provides a brief overview of *LinkWay*. The next two chapters provide simple material that is often covered in beginning books about *LinkWay* and in beginning workshops or training sessions.

Chapters 3 through 20 cover many of the important but relatively easy-to-use ideas about *LinkWay* scripting and some of the tools that come with *LinkWay*. At the same time, these chapters provide an introduction to some computer science concepts. Working through these chapters in sequence, with time allotted to practice new ideas, should provide readers with a good understanding of *LinkWay* scripting.

Chapters 21 through 25 introduce central ideas in programming and computer science: conditionals, looping, scope of variables, and parameter passing. The material in these last few chapters is more difficult for those who have had no previous experience in programming. These chapters would best be used in conjunction with *LinkWay* projects and/or other sources of information about *LinkWay*. The novice programmer should spend considerable time creating new *LinkWay* projects, making use of other sources, and working through the last few chapters of this book.

How to Use This Book

This book is designed so that you can use it while working at the computer. You should read a section and immediately try the ideas discussed in that section. Take time to "play around" with new ideas. Don't simply duplicate the examples in the book. Extend them. Explore. Experiment. If you simply rush from example to example and from chapter to chapter, you will quickly become overwhelmed with new ideas and concepts.

Each chapter of the book includes a summary of the new *LinkWay* words in that chapter. When you get to the end of a chapter, take time to be sure you have worked with each of the new commands and functions. Occasionally the chapter will also contain a section giving keyboard shortcuts. Review these shortcuts from time to time so that you learn to use them to streamline your work. And don't ignore the section in most chapters on debugging and testing. These sections are designed to help you improve your ability to solve problems when working with *LinkWay*.

Note that most chapters have a section on debugging and testing. The term *debugging* is used in computer environments to describe the process of finding "bugs," or errors in your work.

Finally, the book has a number of reference sources to help you find the information you need.

- The Table of Contents lists every chapter and major section in the book.
- The Index includes references to all important concepts as well as a reference to the first use of each new *LinkWay* command or function.
- Appendix 1 presents, by category, all of the keyboard shortcuts given in the book.
- Appendix 2 lists all *LinkWay* commands, including those not discussed in this book.
- Appendix 3 lists all *LinkWay* functions, including those not discussed in this book.
- Appendix 4 lists all *LinkWay* operators.
- Appendix 5 lists the DOS extensions used by *LinkWay*.
- The syntax of each *LinkWay* scripting command or function is given in the text of the book. These can be located by using the index.

These sources can be used to find information when you don't remember exactly where in the text of the book it was introduced. Even after you have worked with *LinkWay* scripting for some time, you will find these references helpful. *LinkWay* includes a large number of special programming words and associated tools. It is unlikely that you will be able to remember how to use all of these new ideas without using these references.

Chapter 0. A Brief Review

This chapter provides a brief review that describes how to use buttons, fields, pictures, pages, and folders in *LinkWay*. It is designed for people who have had some experience with *LinkWay* or who have a great deal of experience learning new software. If you have never used *LinkWay* before or have very little computer experience, you should work with more complete introductory material before beginning this book.

Getting Started

This book assumes that you have had experience using DOS/PC-compatible computers and have had some experience with *LinkWay*. It assumes you have become familiar with the material that comes with *LinkWay* and have created your own folders using pages, pictures, fields, and buttons. It does not teach everything you need to know to get started using *LinkWay*. If you work through the introductory material that comes with *LinkWay*, including the tutorial, and spend some time experimenting with *LinkWay*, you will have sufficient background to begin using this book.

Perhaps you have worked with *LinkWay*, but some weeks or months have passed since that exposure. Perhaps you took a day-long workshop or worked with a friend on a *LinkWay* project. This initial chapter provides the kind of brief review you need to be ready for the material on scripting in Chapter 1. If you currently feel quite comfortable using *LinkWay*'s pull-down menus and the features they contain, you may want to skip immediately to Chapter 1.

What Is *LinkWay*?

If you ask several people what *LinkWay* is, you are likely to get several different answers. Some people see *LinkWay* as an authoring system that can be used to write computer-assisted instruction or hypertext documents. Others think of *LinkWay* as a presentation tool. Some think of *LinkWay* as a tool to control other media, such as CD-ROM drives and videodisc players. Still others think of *LinkWay* as a tool they can use to keep track of appointments, make schedules, and file their addresses.

Why is there such variety in the way people think about *LinkWay*? No doubt the differing viewpoints occur because *LinkWay* is such a flexible computer application. It does not fit nicely into our standard categorizations of tool-type software: word processors, databases,

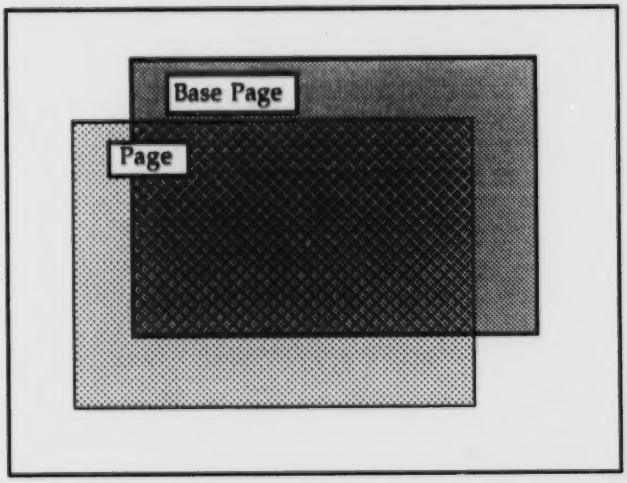
spreadsheets, telecommunications, and graphics packages. It is some of each of these and more.

What, then, indeed, is *LinkWay*? Step back for a moment and look at what components are available when working with *LinkWay*. Each application you use on a computer produces some sort of document. A spreadsheet application produces a spreadsheet document; a graphics application produces a picture document of some kind. In *LinkWay*, the documents are called **folders**. Whenever you open a *LinkWay* document, you have opened a folder.

What is in a folder? The first and most obvious component of a folder is a **page**. You can think of a *LinkWay* folder as being much like a file folder containing a number of pages of paper. Every *LinkWay* folder contains one or more electronic page. Pages are the building blocks out of which you create folders. Information in *LinkWay* is stored on the pages that make up a folder, just as information is written or printed on paper pages stored in a file folder or notebook.

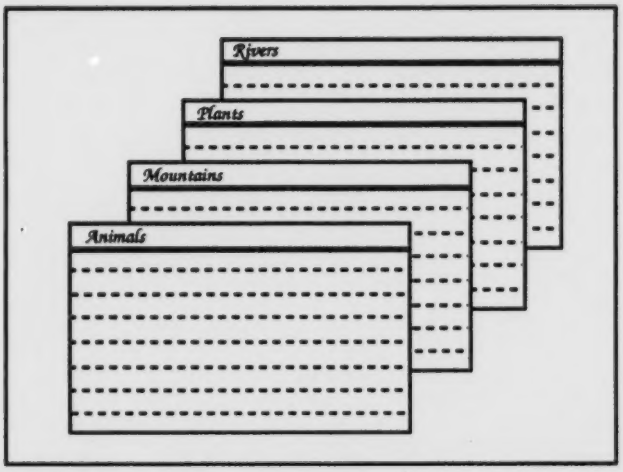
To make this model of *LinkWay* pages more accurate, think of each page (except the base page) as being made of clear plastic or glass. You can write directly on the page—plastic—and you can "glue" things on the page, but at the same time you can see through the page.

Why should you think of the page as transparent? Because each page in *LinkWay* has a background. That background is created on the **base page**. Think of the base page as opaque. You can see the base page through the page, so whatever is put on the base page is visible through the page (unless you put something on the page that covers a spot on the base page):



Yet another way to think about pages and base pages is to think of something familiar, like the kind of index cards that have lines on them.

The lined index cards you buy are all alike—often they have a double pink line at the top and light blue lines on the rest of the card. Your packet of brand-new cards all have the same background. When you use your index cards, you write different things on each card. Although the background of the cards (the printed lines) remains the same, each card looks different because you have put writing on the cards that is on top of the predrawn background lines:



A *LinkWay* folder contains groups of pages all having the same base page, in much the same way that the index cards shown in the previous illustration all have the same background. In *LinkWay* you write different things on each page just as you write different things on each index card.

LinkWay provides tools for "painting" boxes and bars directly on the pages and the base page. It also provides a utility called *LinkWay Paint* that allows you to create graphics that you can import as pictures into *LinkWay*. In addition, you can add objects called buttons and fields to base pages and pages. Thus you can have fields, pictures, and buttons on the base page as well as fields, pictures, and buttons on the page. A more complete review of buttons and fields is given later in this chapter.

In *LinkWay*, fields, pictures, and buttons are all called objects. In *LinkWay Live!*, you also have objects called "media" and "list." List objects will be discussed later in this book. Media objects are discussed briefly in Appendix 6. Everything you create on a page in *LinkWay* is done by using these objects; everything you see on pages in folders created by others is accomplished by using these objects. Sometimes you will hear people talking about *LinkWay* as an "object-oriented" environment. It would be more accurate to call *LinkWay* an "object-like" environment. True object-oriented languages allow you to create new types of objects; *LinkWay* allows you to create only the kinds of objects previously described. While *LinkWay* retains some of the characteristics of an object-oriented

environment, it does not have the full power of an object-oriented programming language.

What You Need to Know to Begin

This book assumes you have either worked with *LinkWay* in the "immediate mode" or you have programming experience and understand how to use a DOS/PC-compatible computer. That is, it assumes that you have either spent some time with *LinkWay* becoming familiar with the tutorial and have created folders, pages, pictures, fields, and buttons, or that you know a good deal about programming languages.

The concepts and techniques you should understand to continue with this book are reviewed briefly in the following sections. If you find them difficult to follow, spend some time with a beginning *LinkWay* book, such as *LinkWay for Educators*, until you are comfortable using pull-down menus and the mouse to create simple folders containing buttons, fields, pictures, and pages.

A Brief Review—Creating a Folder

If *LinkWay* is installed on the fixed disk in your system, recall that to start *LinkWay Live!* you type

```
llgo
```

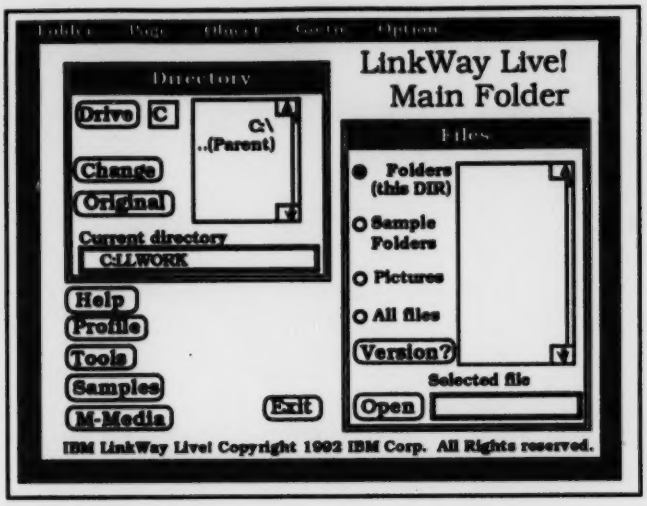
For more details on startup options for LinkWay Live!, *see the documentation that comes with your software.*

To start *LinkWay* on a fixed disk system, you type

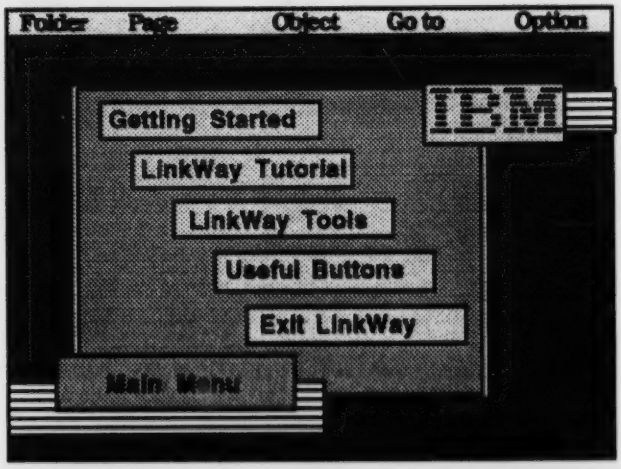
```
lwgo
```

If you are using LinkWay *on a disk-drive system, see your* LinkWay *documentation to learn how to start the software on your computer.*

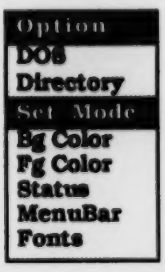
If you are using *LinkWay Live!*, you see



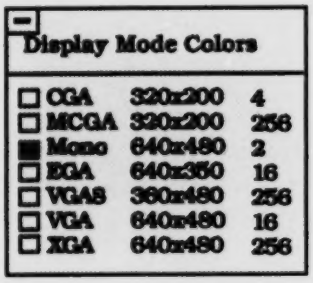
If you are using *LinkWay*, you see



Before you create a *LinkWay* folder it is a good idea to set the mode. Point to **Option** on the menu bar. Click either mouse button. Move the mouse so that **Set Mode** is highlighted and again click either mouse button.



You then see a window that looks something like this.



See the documentation that comes with LinkWay *for more information on graphics modes.*

Select the display format you wish to use by clicking in one of the boxes.

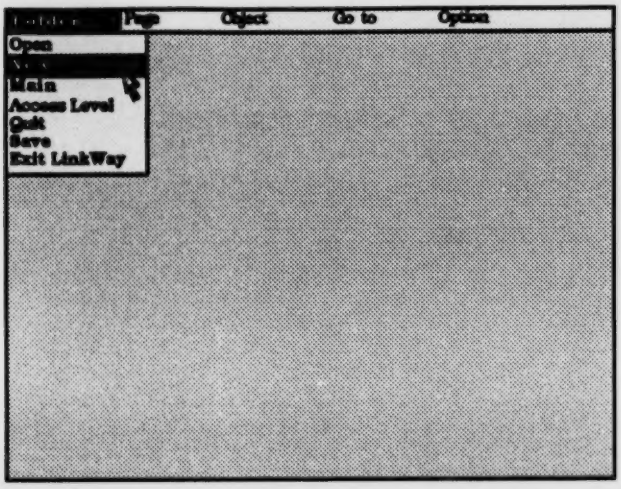
Because *LinkWay Scripting—An Introduction* is printed in black and white, screens shown in the book will be resemble screens in which the computer is set in Mono display mode.

Note that a folder created in one graphics format may only be opened in that same display mode. See your LinkWay *manual for more details.*

Recall that the small "–" in the upper left corner of the window is called the "Close Dot." Put the mouse pointer on the Close Dot (⊟) and click. The window disappears.

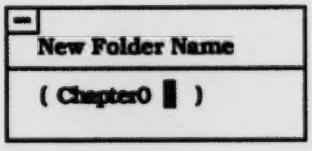
In LinkWay, *clicking on the Close Dot will always cause the window to disappear. However, sometimes pressing the Enter key will also cause the window to disappear. In addition, you can remove any window from the screen by clicking outside of that window. Even though there are often several ways to remove a window, this book will always tell you to use the Close Dot. However, you can use any method you wish to close windows.*

Now you are ready to create a new folder. Choose the **Folder** menu and select **New**.

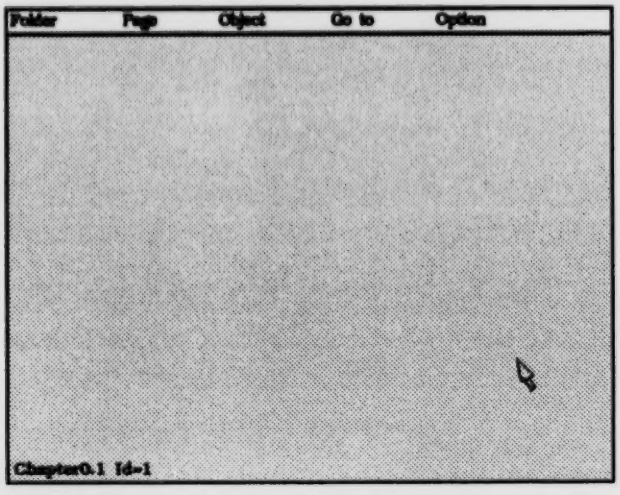


Note that pages, menus, and windows will be shown the way they appear in LinkWay Live! *in monochrome mode. Your screen may look somewhat different, but the choices should be nearly the same.*

You will then be asked to name your folder. You might name this new folder Chapter0.

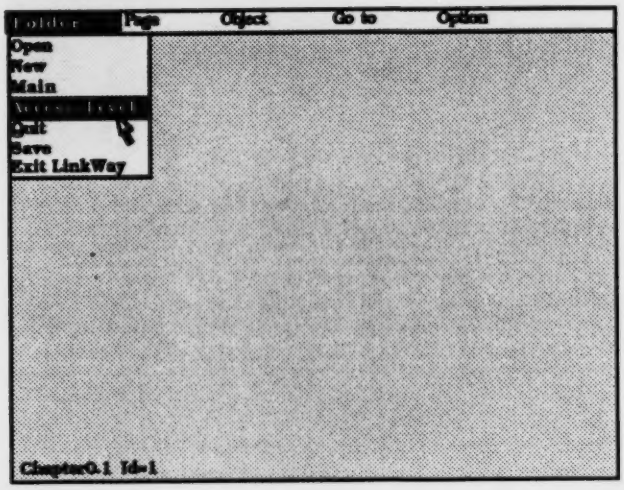


The name of the folder, the number of the page, and the Id number of the page appear at the bottom left corner of the page in the Status Line.

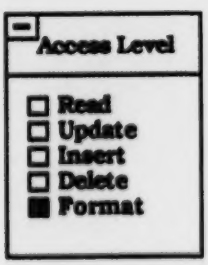


If you are using *LinkWay* rather that *LinkWay Live!*, you will see **Chapter0.base** in the lower left corner of the page.

Before working with your new folder, however, you need to be sure your Access Level is set correctly. Click on the **Folder** menu on the Menu bar at the top of the page. You see the submenu under **Folder.** Move the mouse so that **Access Level** is highlighted.



Set the Access Level to **Format**



Recall that to create and modify folders, you need to have the Access Level set to Format. The other access levels restrict the changes that can be made in a folder. See your *LinkWay* documentation for more details on the various Access Levels.

A Brief Review—Creating a New Page

When you create a folder, it already contains two pages, the base page and the page identified with the name of the folder, the page number, and the Id number.

If you are using LinkWay *rather than* LinkWay Live!, *only a base page is automatically created. You must use* ***New Page*** *to create the first page of the folder.*

You can create a new page by choosing **New** from the **Page** menu. The new page looks identical to the first page. The only difference is that the second page has

```
Chapter0.2 Id=2
```

at the bottom of the page. You now have two pages. The number after Chapter0 is the page number.

Note that if you are using LinkWay *you will have created two new pages at this point.*

Take a moment to check the contents of your folder. Point to the **Go to** menu and choose **Base page.** You should see the base page. Now use the **Go to** menu again to go to the **Next** page twice. The identifying information at the bottom of the page should change.

Note that the page number—the number after the name—indicates the order of the pages in the folder. The Id number represents the order in which the pages are *created*. For the two pages created above, the page number and the Id number are the same. That will not always be the case.

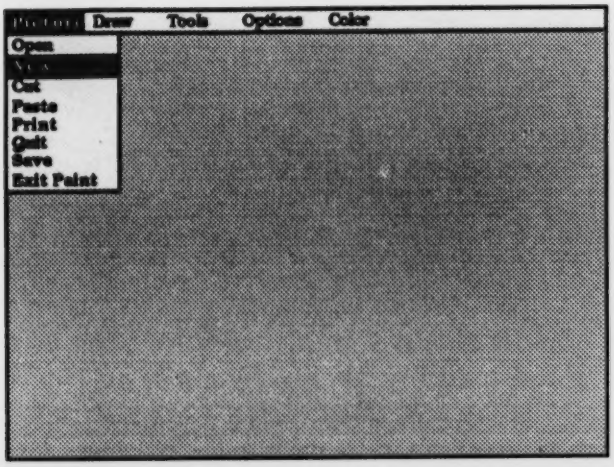
A Brief Review—Using *LinkWay Paint*

To tell the difference between the two pages you have created, you can add a picture to each one. Go to the first page. Under **Object** on the Menu bar, select **Tools**. You see the Execute Tool window.



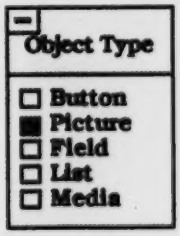
Click on the downward pointing arrow until you see LWPAINT. Click on LWPAINT and click the Close Dot. Reply "Yes" when asked whether to start the LWPaint program.

When the *LinkWay Paint* Menu bar appears, select **New** from the **Picture** menu.



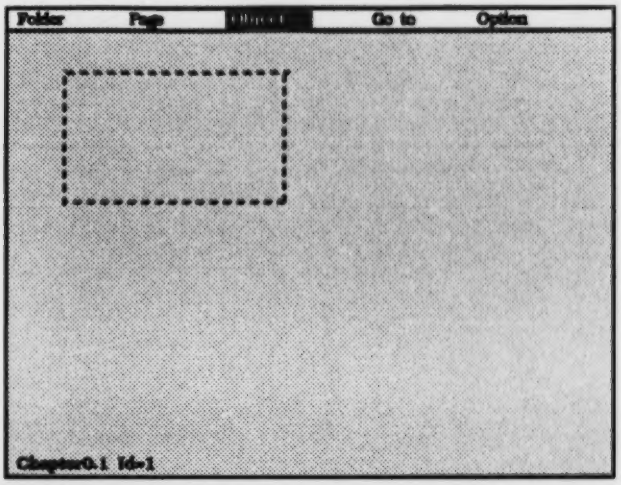
Name the picture "First." Then select **Text** from the **Draw** menu. A small, rectangular cursor appears. Type the words "My first page" and then click the mouse on the left side of the screen to "glue" the text to the picture. Note that the text that you put on the page in this manner is actually "painted" onto the picture. That is, the letters behave like drawings, not like word-processed text. Text created in *LinkWay Paint* actually behaves like a picture.

Select **Exit Paint** from the **Picture** menu. You automatically return to the first page of your folder. Click on **Object** on the Menu bar and select **New**. Because Pictures are objects in *LinkWay*, you see



Note that if you are using LinkWay, *you will not have the List and Media choices.*

Click in the box next to Picture and then click the Close Dot. A small dotted rectangle appears. This rectangle is called the *bounding box*. Click at the upper left corner of the box. Move the mouse until the bounding box covers the part of the screen on which you want your picture.



Click the mouse a second time. You are then asked for the name of your picture. This is the name that will be used in your folder, *not* the name used to save the picture in *LinkWay Paint*. You might name this picture "Pic_P1."

> *Note that an underscore character (_) can be used in the name of a picture or any other* LinkWay *object. Because it is advisable not to have spaces in the names of objects, the underscore character can be used to make the name of an object more readable.*

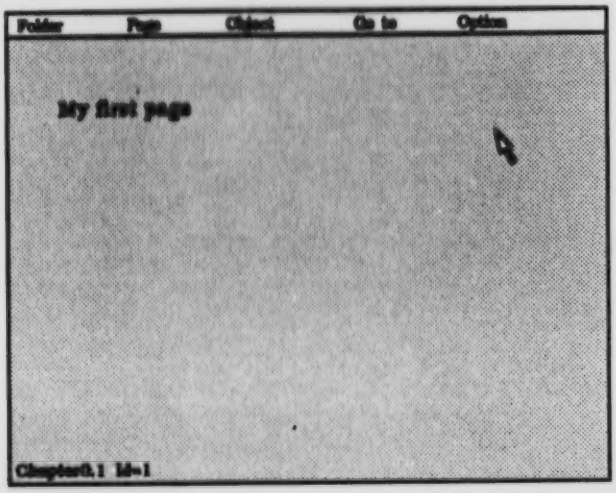
Click the Close Dot.

> *Note that picture objects are opaque; that is, the bounding box for a picture will hide anything that is under it on the base page.*

Next you see a list of available pictures. Select "First"—or the name you used to save your picture. Click the Close Dot.

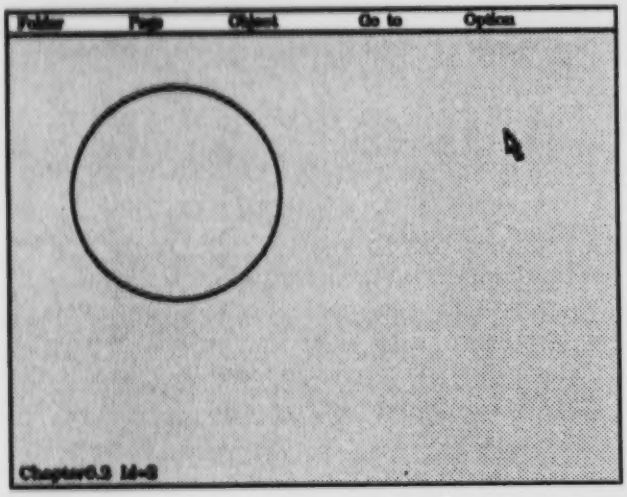
> *If you want to have a blank picture object, leave the name of the picture blank. An opaque area the size of the bounding box will then be placed on the screen. Opaque picture objects can be used to cover objects on the base page that you don't want to be visible.*

The picture should appear on the left side of the page. The Adjust window appears. Click on FullPic at the bottom of the Adjust window so that you can see the entire picture object. You can then use the Up, Down, Left, and Right options in the Adjust window to move the picture into the bounding box. Your first page should now look like this.



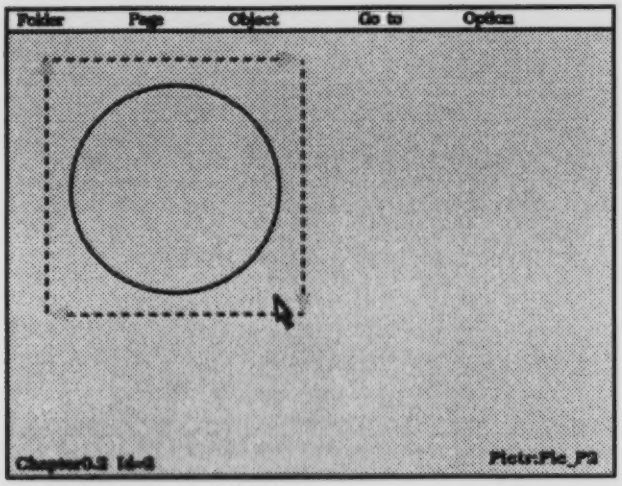
Note that even though there is text on the page, it is still called a picture object. Text that is part of a picture cannot be edited unless you return to *LinkWay Paint* and use the graphics tools available there. It is important to understand the difference between text you can edit—like the text typed in a word processor—and text created by a graphics program.

Now go to the second page of your folder. Exit *LinkWay* and go to *LinkWay Paint*. Select **Circle** from the **Draw** menu and draw a circle on the left side of the screen. Name your picture "Circle." Return to *LinkWay*, create a picture object called "Pic_P2" and put the circle picture on the left side of the second page in your folder.



Now you can easily tell which page you are on as you move through your new folder. Note that if you select **Next** from the **Go to** menu when you are on the second page, you will be moved to the first page.

Note that if you click on an object, its bounding box appears and its name appears in the Status Line in the lower right corner of the page.

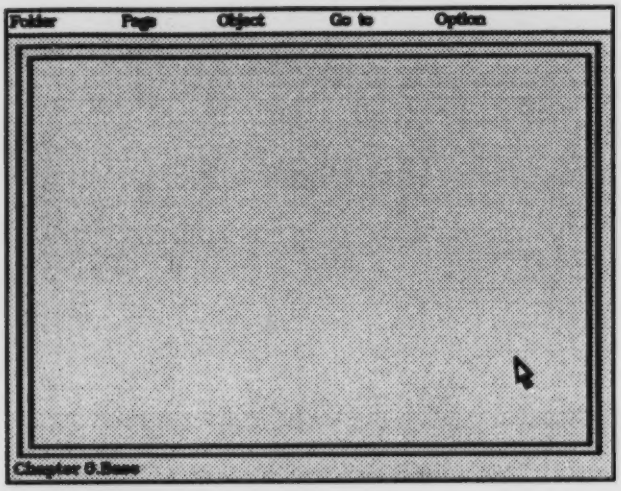


When you are working with pictures, the name that appears on the Status Line is the name of the *picture object*, not the name of the *LinkWay Paint* file containing the picture. Be sure you understand this distinction.

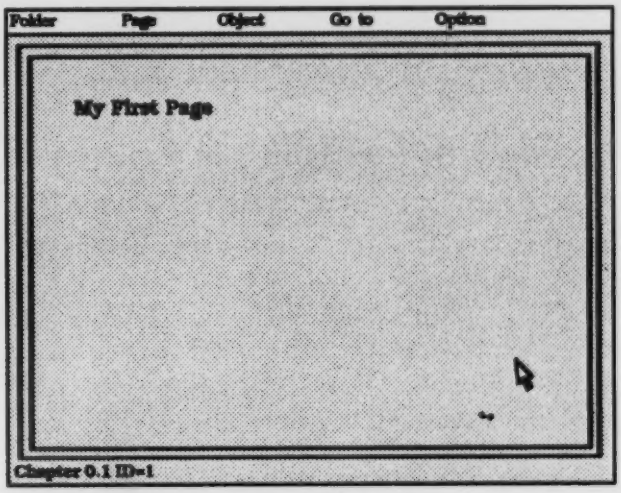
A Brief Review—Editing a Base Page

You now have two pages in your folder. You also have a base page that you have not modified in any way. Select **Base page** from the **Go to** menu. You see **Chapter0.Base** at the bottom left of the screen. You do not see either the text picture or the circle that are on your first two pages. Now, you are going to modify the base page.

Select **Box** from the **Page** menu. Draw a frame around the edge of the base page. Draw a second frame inside the first frame.



Now select **First Page** from the **Go to** menu. You see your text picture "over" the new base page:



Examine the second page in your folder. You see a circle "over" the frame on the base page. The base page appears to be "behind" all of the pages in your folder.

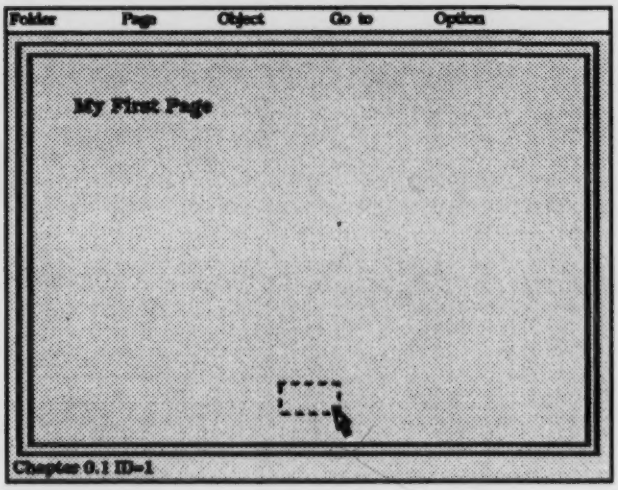
> *In* LinkWay *you can have only one base page per folder. If you need different backgrounds, you must create several folders and link those folders together.*

You can modify your base page at any time without affecting any of the pages in your folder. Keep in mind, however, that changes made on the base page will be seen on every page in your folder.

Objects on the base page may be edited *only* on the base page. If you are on the base page and click on an object, the name appears in the status line on the right side of the page. However, if you click on a base page object while on another page, the object name appears with a ".B" after its name.

A Brief Review—Creating a Button

Buttons are commonly used to link pages so that you can easily move from one page to another without using the **Go to** menu. Begin your review of working with buttons by creating a button to move to the next page in the folder. Go to your first page. Choose **New** from the **Object** menu and check Button. A bounding box appears. Place the button at the middle of the bottom of the page.



When the Button Type window appears, check Go. Next, you are asked for the Button Name. Type "Next_Pg" for the name and click the Close Dot. It is always a good idea to name your buttons. It will help you identify those buttons at a later time when you have many objects on your pages.

Next, the Button Icon window appears. There are four forms in which a button can appear. If you select a picture from the Button Icon window, the Button will appear as an *icon*. An icon is a small picture that represents the function of the button.

The target area—the part of the button that responds to a mouse click—for a button with an icon can be in any of the four corners of the bounding box depending on the way in which the bounding box is drawn. That is, if the bounding box is drawn from the upper left, the icon and target area will be in the upper left. If the bounding box is drawn from the bottom right, the icon and target area will be in the lower right corner.

If you select the name of the button, the name will appear on the button. The word "sign" is used to refer to a button on which the name appears. The sign can be one of three different shapes.

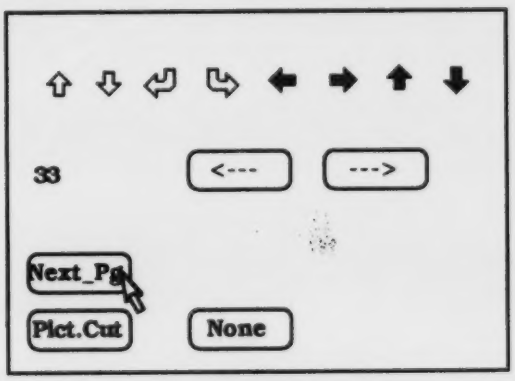
In LinkWay *there is only one button shape available.*

The target area for a button with a sign is the shape containing the name of the button.

If you select None from the Button Icon window, the button will be invisible and the entire bounding box is the target area. This type of button can be used to make any area of the screen selectable, for example, various areas of a picture.

The fourth choice for the form of a button is a Picture Cut button. Picture Cuts will be discussed in Chapter 19.

For this button to link to the next page, click on the button name so that it will have a sign.



Note that if you are using LinkWay *rather than* LinkWay Live!, *the Button Icon window will look slightly different.*

Finally, choose **Next** from the **Go to** menu. The button appears at the bottom of the page.

"Next" buttons always go to the page after the current page in the folder. If you are on the last page of a folder, a next button goes to the first page. Similarly, "previous" buttons go to the page just before the current page. If you are on the first page of the folder, a previous button takes you to the last page of the folder.

If you are not satisfied with your button, you can make changes. Click on **Object** on the Menu bar. You use **Edit** to change the characteristics of the button. **Move** is used to put the button in a different place. **Move + Size** is used when you want to move the button *and* change its size.

When you are creating or moving objects, they automatically conform to a coordinate grid. You can force the objects to conform to the character-cell grid by pressing and holding down the Alt key while drawing the

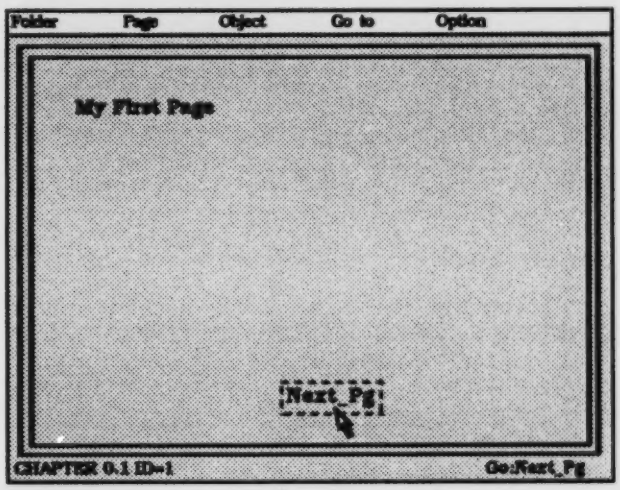
object's bounding box. Spend a few minutes reviewing how to edit, move, and resize buttons. Try using the Alt key to place objects.

> *Note that the area in which you can click to select or activate any object is called the* object target area. *The object target area is not the same as the bounding box. In the button just created, the object target area is the round rectangle containing the name of the button.*

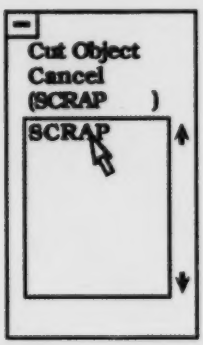
Perhaps you would like to have this Next_Pg button on every page. Be sure you are on page 1. Click on the button. A bounding box appears and you see

```
Go:Next_Pg
```

in the bottom right corner of the page.



Go to the **Object** menu and select **Cut.** You see the Cut Object window.



You probably want to cut the object and put it in Scrap, so click on Scrap and click the Close Dot. (If you wish, you can use another name for temporarily storing the object you cut.) You should then see a small window that says "Cut completed." Click on the Close Dot.

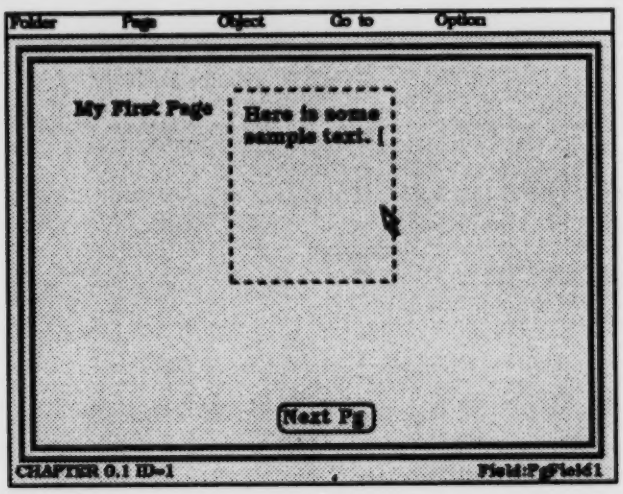
Next, go to the second page in your folder. Select **Paste** from the **Object** menu. The Paste Object window appears. Click on Scrap—or the name you used to save the object. A dotted rectangle appears. Place the rectangle in the center of the page at the bottom. When you click the mouse button, the button appears.

Now, test your buttons. Click twice rapidly (double click) on the Next_Pg button. You go to the next page. You can now cycle through your two pages with ease.

Note that when you are in the Format Access level, you must double click on a button to activate it. In *all* other modes, you only click once to activate a button.

A Brief Review—Creating a Field

To create a field, you follow steps similar to those you used when you created a button. Again begin on the first page. First choose **New** from the **Object** menu and put a check next to Field. A bounding box appears. Place the field in the center of the first page.

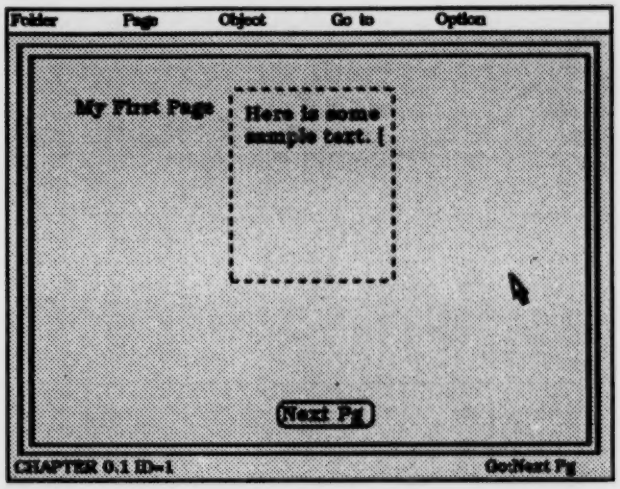


Next, choose a font. You can select from each of four fonts. These are

- headline type
- type for text
- type for special uses
- boldface type

Next, name your field, perhaps PgField1. If you are not satisfied with the placement of your field, use either **Move** or **Move + Size** from the **Object** menu.

Now move the arrow pointer over the field. Double click. A left bracket cursor ([ ) appears. Type the words shown in the following illustration.



Click outside of the field, and the bounding box of the field disappears.

There are actually two ways you can enter text. When you see the left bracket cursor ([ ), you are in Insert mode. In this mode, words automatically wrap to the next line. If you press the Ins (or Insert) key, the cursor changes to a vertical bar (|) and the letters typed replace the letters that are already there. You are in Replace Mode. Words do not *automatically wrap to the next line. Press the Ins key again and you will again be in Insert mode.*

Recall that the text you created earlier using *LinkWay Paint* is "paint text." Such text is part of the graphics in a picture. It can be changed only by using painting tools in *LinkWay Paint*. The text you entered in the field is text that can be edited like the text created when you use a word processor. Try placing the arrow in the middle of the line you just typed. Click twice. The [cursor appears. (If you see a | cursor, press the Ins key once.) Any letters or words you type are inserted at that point. You see that you can easily modify the text in a field.

You now have a folder that contains two pages, two buttons, and a field. Have you been saving your work? It is a good idea to select **Save** from the **Folder** menu every 10 or 15 minutes. In any case, when you quit, *LinkWay* will store any changes you have made, so they will always be available next time you use the folder.

If you are not yet comfortable with the material in this chapter, you should spend some more time experimenting with creating your own *LinkWay* folders. If you are comfortable with the material reviewed so far,

you are ready to move on to the next chapter, where you will begin learning about scripting in *LinkWay*.

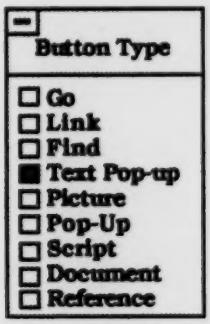
Chapter 1. Buttons and Scripting

In this chapter, you will learn to create buttons that allow you to associate either text or a picture with the button. In addition, you will learn to write simple button scripts.

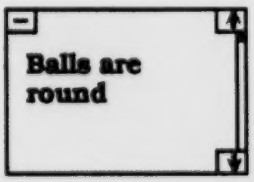
Using Pop-Up Buttons

LinkWay provides two kinds of buttons that are quite useful for temporarily displaying information. These two kind of buttons, Text Pop-up and Picture Pop-up, make it easy to have buttons display supplementary information.

Begin by creating a new folder. You might want to call it Chapter1 to help keep track of your experimental folders. Create a new button of type Text Pop-up.



Name this button Round. A text field appears. Double click in the window so you see the [cursor. Type some text into the window. For example, you might type "Balls are round."

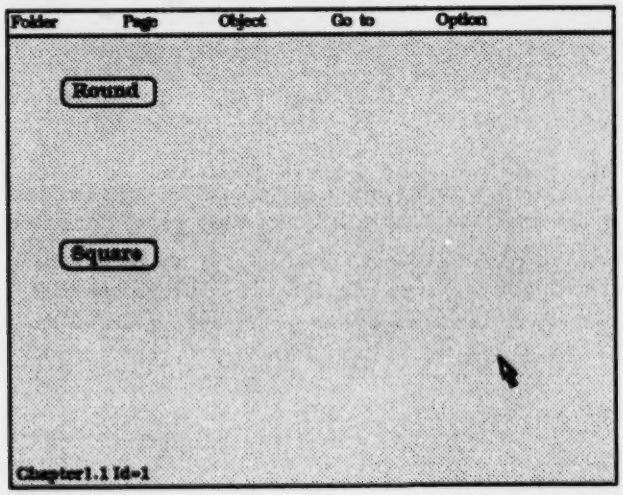


Note that the size of the pop-up window is the same size as the bounding box for the button. If you want a larger window into which you can type information, use **Move + Size** from the **Object** menu to make the button bounding box larger.

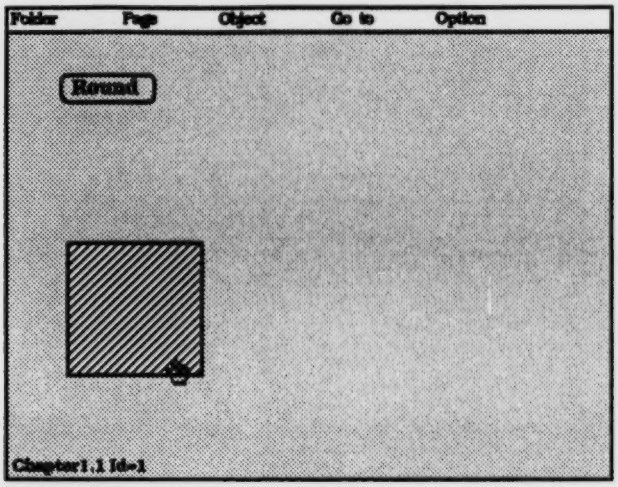
Now click on the Round button. The Pop-up window appears. Click on the Close Dot and it disappears. The button is visible again. You have created a button that is automatically "linked" to some explanatory text.

> Note that you are most likely working in Format Access level. To activate buttons when working at the Format level, you must double click on them. In Chapter 0 you were reminded to double click. However, from this point on in this book, the word "click" will be used when you are asked to activate a button, even though you may need to double click.

Next create a Picture Pop-up button. Begin by going to *LinkWay Paint* and creating a picture. Make a picture of something square. You might just want to make a simple colored square. Return to *LinkWay* and create a Picture Pop-up button called Square. Be sure you make the bounding box for the Square button big enough to hold your picture. The first page of your folder should look something like this.



If you double click on the Square button, you see something like this:



When you click again, the button reappears.

Pop-up buttons can be quite useful for displaying more information than can easily fit on a single page. For example, if you are creating a page for young children, you probably should use Picture Pop-up buttons. On the other hand, if you are creating a page that has buttons for technical terms, you should probably use Text Pop-up buttons to display definitions.

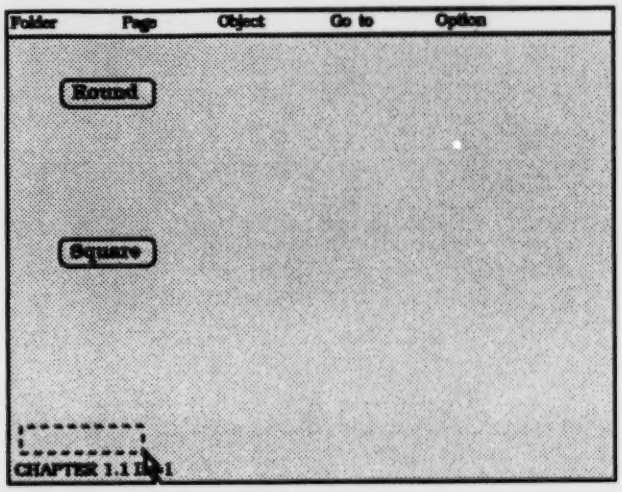
Introduction to Scripting

LinkWay makes it easy to create folders of many pages. These pages can contain buttons, fields, and pictures, and the pages can be linked together. However, to fully use the power of *LinkWay*, you need to learn about *scripting*. Scripting is the word used to describe writing a program in the programming language built into *LinkWay*

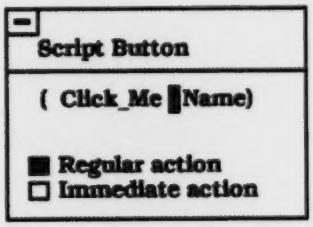
To begin exploring how to write simple scripts, add a new page to your Chapter1 folder. Note that the Status line in the lower left corner of the page says

```
Chapter1.2 Id=2
```

Use the **Go to** menu to return to the first page of your folder. Create a new button and place it just above the Status line.



Select Script from the Button Type window. Give the button a name, perhaps Click_Me! (Note that Regular action should already be checked in the Script Button window.)



Choose Click_Me from the Button Icon window. The Script window appears. Type

```
go 2;
```



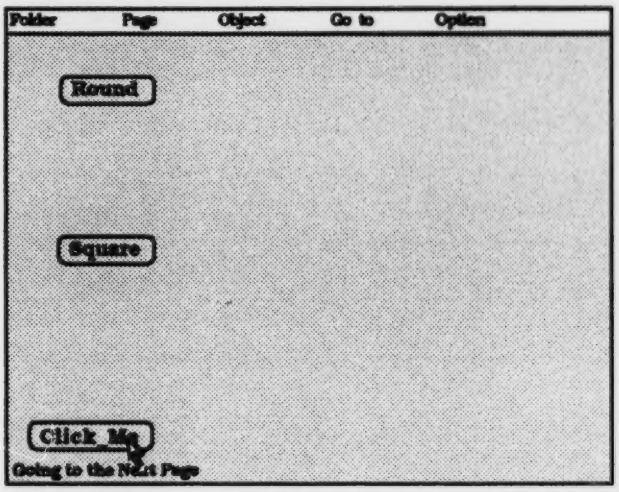
and click the Close Dot.

Now click on the button. You go to the second page. The **go** command is used to go to a specific page in a folder. The number after the **go** command is the number of the page—it is not the Id number. The semicolon at the end of the statement is used at the end of all *LinkWay* scripting statements. Although lowercase was used in this statement, you can use uppercase or mixed case as well.

Next, select **Edit** from the **Object** menu. When you get to the Script window, add a **prompt** command and a **wait** command so that the script looks like this:

```
prompt "Going to the Next Page";
wait 2;
go 2;
```

Now when you double click on the Click_Me button, the **prompt** command puts "Going to the Next Page" in the Status line at the bottom left corner of the page.



The **wait** command causes *LinkWay* to pause so that you have time to read the Status line. The number after **wait** is the number of seconds that *LinkWay* pauses.

Working with this new script can be frustrating because it moves you away from the first page of the folder each time you activate the Click_Me button. You may find yourself constantly having to move back to your first page to experiment some more. Rather than use **go** for experimenting with scripts, you might consider using the **beep** command. Thus, you might change the previous script so that it reads

```
beep;
beep;
beep;
```

Now each time you double click on the button, three beeps occur and you remain on the same page.

As indicated previously in this book, *LinkWay* has a programming language that is part of the *LinkWay* application. Programming languages have a structure and set of rules that define the correct way to use them, just as English grammar rules define how to use English. As you progress through this book, you will learn the general rules—or *syntax*—for using each new command or function. In Chapter 7 you will learn more about the general structure of the *LinkWay* scripting language. It is important that you progress beyond memorizing and using specific examples of *LinkWay* scripts. You should make an effort to understand the structure of each new command or function as it is introduced.

In this chapter you have seen several commands. The correct way to use the **beep** command—its syntax—is given in the following box. (Note that the syntax of commands will be shown in plain rectangular boxes like the following one.)

```
beep
```

The semicolon (;) that comes at the end of each statement is not shown. To get several beeps, you must use several beep commands separated by semicolons.

The **go** command has the syntax

```
go pagenumber
```

Here you see that **go** has one *input*. That input must be a number that represents the number of a page. Note that **beep** does not have an input.

Your LinkWay *documentation uses the word "operand" for inputs to commands. Another word often used for inputs to commands is "parameter." However, in this book, the word "input" will be used to describe the values that a command or function needs.*

The **wait** command also has one input. This input is the number of seconds that *LinkWay* pauses. Note that you can use decimal values for seconds.

Thus the syntax of **wait** is

```
wait seconds
```

When you see italics in a syntax description, you substitute a value for the italicized word when you use the command.

The command **prompt** takes an input. It must be followed by a string of characters enclosed in quotation marks, called a *quoted string*.

```
prompt expression
```

Prompt must be followed by a message to be displayed on the Status line. *Expression* is a general term for the kinds of information that can go after **prompt**. You will often see *expression* in syntax diagrams. In this example, the expression is a quoted string of characters.

Spend some time experimenting with simple scripts. For example, make a Script button to replace the Square Pop-up button. Have it beep four times and display the message "A square has four corners." You could make similar triangle, pentagon, and octagon buttons.

Debugging and Testing

As you begin your work with *LinkWay*, it is easy to get lost or confused as you create more and more pages. Keep in mind that you can always use Status line to give you information about your location. The Status line gives the name of your folder, the number of the page in the folder, and the Id number. The page *number* tells you the sequence of the pages in your folder. Thus, if the page number is 10, you are on the tenth page of your folder. The Id number is assigned by *LinkWay*. The Id number reflects the order in which the pages were created. Thus, if you insert a page between the first and second page of a two-page folder, the new page will have page number 2 and an Id number of 3. The last page of the folder then has page number 3, but an Id of 2 because it was created before the middle page.

When you begin scripting, you must be very careful about typing commands. A simple typographical error will cause your script to work incorrectly. Check your scripts carefully before you leave the scripting window.

Although it is not always necessary to have a semicolon after every *LinkWay* scripting command, using a semicolon is a good habit to develop now. As your scripts become more complex, good scripting style will help your debugging efforts.

New *LinkWay* Words in This Chapter

Commands
`beep`
`go pagenumber`
`prompt expression`
`wait seconds`

Project Ideas

Spend some time experimenting with scripting buttons to produce a variety of messages and sounds. You can mix **wait** commands with **beep** commands to get interesting rhythms. You can use **beep** to call attention to messages displayed with **prompt**.

Create a folder containing five or six pages. Use the **go** command to link these pages. For example, you could begin with the Square and Circle buttons you made previously in this chapter. Then you can have a page full of circles, a page with text giving examples of circles, and a third page with a definition of circles. You can have a similar set of pages for the Square button. Then use **go** to link these pages in a meaningful way. Use **wait** and **beep** in the transitions from page to page.

Chapter 2. Displaying Text and More

In this chapter, you will learn about putting text into fields by using scripting commands. In addition you will learn about functions.

In the last chapter you learned how to write *LinkWay* scripts. These scripts were placed on a kind of button called a Script button, which caused some action to occur when you clicked on the button. You also learned to use the **prompt** command to put text on the screen. In this chapter you will learn more about displaying text by using *LinkWay* scripting.

The Prompt Command

You have learned to use the **prompt** command to put short messages in the Status line at the lower left corner of the screen. Recall that the syntax of the **prompt** command is:

```
prompt expression
```

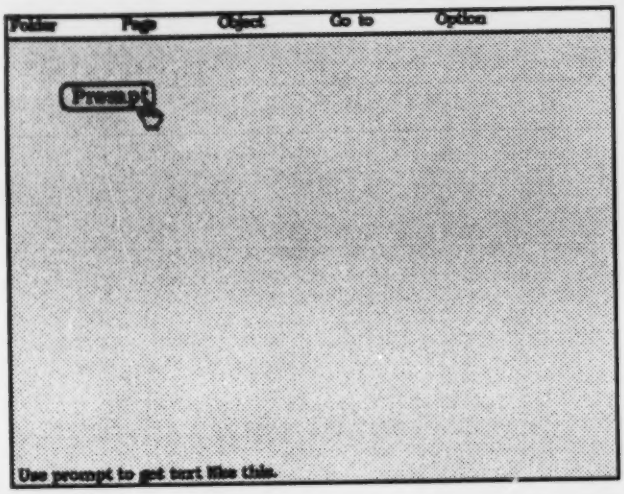
Expression is the value—the input—that must be used with **prompt**. *Expression* contains the characters that are to appear at the bottom of the page.

To begin examining and comparing ways to put text on the screen, create a new folder. Perhaps you want to call it TextEx because you will use it to explore a variety of ways to put text on the screen. Or perhaps you want to call it Chapter2 to keep track of your work in this book.

Begin by creating a Script button called Prompt. Put the button in the upper left corner of the page and select the sign from the Button Icon window so that the name of the button appears on the button. Then create a sample script using **prompt**. For example, you might use

```
prompt "Use prompt to get text like this.";
wait 2;
prompt "";
wait 1;
prompt "Use prompt to get text like this.";
```

The first **prompt** command displays the text. Then there is a two-second pause. Then the "empty" string after the second **prompt** statement makes the text go away. Finally, after a one-second pause, the text reappears.



Prompt is easy to use but very limited. You can only put a few words after **prompt,** and the words will always appear in the lower left corner of the screen in the Status line. However, **prompt** can be useful for displaying reminders to those using your folder. The text can remind the user what to do next or display information about what is currently happening. You can easily design all of the pages in your folder so that the information you put on the page will not interfere with anything on the Status line.

The Msg Command

Another way to display text on the screen is to use the **msg** command. **Msg** puts text on the screen in a window. The syntax of **msg** is:

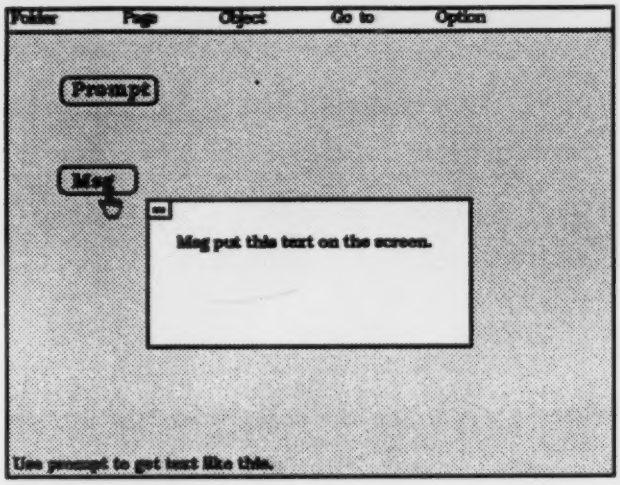
```
msg expression,[expression];
```

The square brackets ([ ]) are something you haven't seen before in syntax diagrams. Inputs that appear in brackets are optional. In the case of **msg** this means that either one string of text or two strings of text may be given as input to **msg.**

To explore the use of **msg** , create a Script button and call it Msg. Put a sample script using **msg** on it. For example, you might use

```
msg "Msg put this text on the screen.";
```

When you click on the Msg button, you see something like this:



Experiment some more with **msg**. What happens if you use two inputs? That is, what happens if you use

```
msg "Msg put this text on the screen.","Here is a
  second line.";
```

Try different lengths of text strings after **msg**. What happens?

If you are using *LinkWay Live!*, you can place the **msg** window at a specific location on the screen. Modify the above script so that it reads

```
drowcol 10,30;
msg "Msg put this text on the screen.";
```

When you click on the button, the message window moves diagonally to the right and part way down the screen.

The placement on the screen depends on the mode in which you are working.

Drowcol stands for "draw row column." Its syntax is:

```
drowcol rownumber,columnnumber
```

The first input is the number of character rows *down* the screen the message window will appear. The second input is the number of character columns *across* the screen that the message window will appear. Note that this reference to row and then column is exactly the opposite of the x- and y-coordinate order you learned in math classes.

Note also that the range of rows and columns varies with the graphics mode you are using. The range of rows may be 0-24 or 0-29, while the range of columns may be 0-39, 0-44, or 0-79. See your LinkWay *documentation for details.*

Now try modifying the previous script. Try reversing the row and column inputs.

```
drowcol 30,10;
msg "Msg put this text on the screen.";
```

This time the message window appears near the bottom of the screen. Note that

- the row and column numbers you provide are situated at the upper left corner of the message window.
- if the numbers you specify are too big, *LinkWay* will place the window so that it fits on the screen.
- setting either the row or the column to zero puts the message window in the default position in the middle of the screen.

Spend some time experimenting with **drowcol**. What are the largest values you can use with your current mode setting? If you have access to several modes, explore the differences among modes.

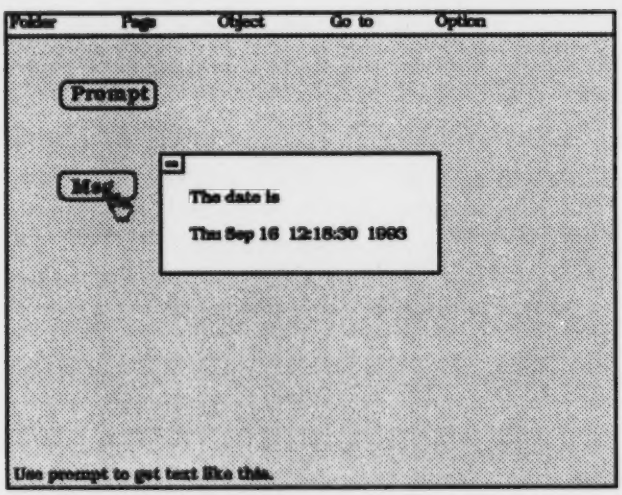
Functions

In addition to *LinkWay* scripting commands, there are also *LinkWay* scripting *functions*. *LinkWay* scripting commands cause some action to occur, such as moving to another page or displaying a message. Functions produce values. An example of a function you might find useful is the **time** function.

Modify the previous script so that it reads

```
msg "The date is ", time;
```

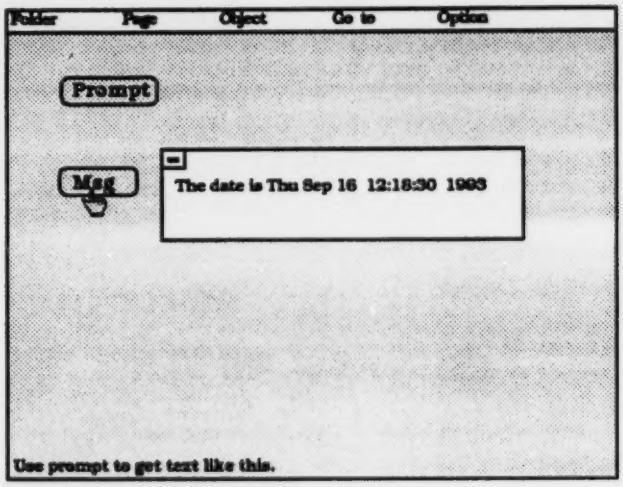
When you click on the Msg button you see something like this:



If you want to have the text and the time on the same line, you can use a colon (:) instead of a comma. The colon concatenates—puts together—the two strings of text. Thus, modifying the **msg** command as follows

```
msg "The date is ":time;
```

causes the page to look like this:



The syntax of **time** is as shown in the following double box. Recall that commands are shown with a single box around them; functions will be shown inside a double box.

```
time
```

Time always returns a 24-character string that has the following format:

Characters	Meaning
1-2	Day (Alphabetic)
4	Blank
5-7	Month (Alphabetic)
8	Blank
9-10	Day (Numeric)
11	Blank
12-13	Hour
14	:
15-16	Minutes
17	:
18-19	Seconds
20	Blank
21-24	Years

If you are using *LinkWay Live!*, there is another function that deals with time. This function is called **itime**. Its syntax is as simple as the syntax of **time:**

```
itime
```

Itime returns a number in seconds. Make a new button and call it Seconds. Put the following script on the button:

```
prompt itime;
wait 3;
msg itime;
```

You see a number like 73558 on the Status line. There is a pause, and a message window then appears with a different number, for example, 73561. The last digit should have changed by 3 because the script had a **wait 3** in it. This function can be used to time events that take place in your folder.

Using the Text Command to Display Messages

The **text** command is yet another way to display text on the page.

Note that the ***text*** *command is not available in* LinkWay. *Read this section only if you are using* LinkWay Live!

Text has a more complex syntax than any of the commands you have seen before. **Text** gives you more flexibility than the other commands you have seen to display text from a script. By using **text,** you can place text anywhere on the screen and in any color you wish. Here is the syntax for **text:**

```
text color,x1,y1,x2,font,text-expression
```

The first input, color, is a number representing the color of the text. See your *LinkWay* documentation for details on color numbers. For these examples, we will use 15 for white. The next three inputs specify the placement of the text on the screen:

- x1 is the left margin in pixels.
- y1 is the vertical starting position in pixels.
- x2 is the right margin in pixels (x1 must be less than x2).

Pixel is a word that means "dot on the screen." The number of pixels displayed on the screen depends on the mode you are using.

The number of pixels in the screen is quite variable. To learn the pixel ranges on your screen, you can use the Tracker tool (see Chapter 5). Details about the various modes also are given in the appendixes of your LinkWay Live! *manual.*

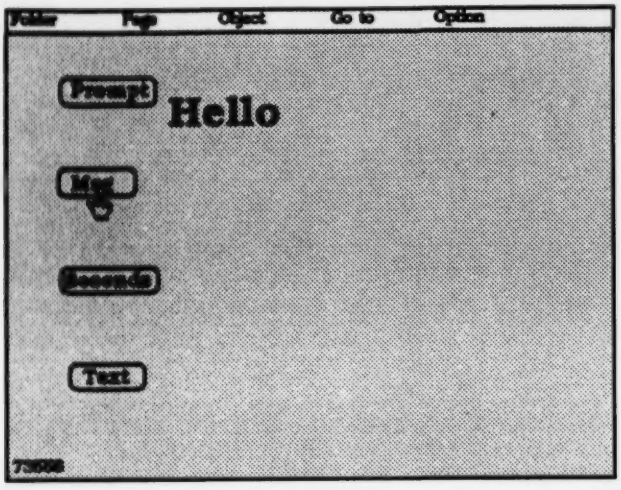
Finally, the fonts can have numbers 0, 1, 2, or 3. The 0 is the top font in the font window for fields.

Note that the text created by the **text** *command cannot be edited after it is placed on the screen.*

Create a new button and call it Text. Put the following script on the button:

```
text 15,150,100,250,3 "Hello";
```

Now try the button. A large "Hello" should appear on the screen.



Keep in mind that the placement of the text on your screen depends on the characters of the screen and mode you are using.

You can create some interesting effects using **text**. Add the following lines to the script on the Text button:

```
text 15,150,100,250,3 "Hello";
wait 1;
text 0,150,100,250,3 "Hello";
wait 1;
text 15,150,100,250,3 "Hello";
```

When you click on the button, the word Hello appears, disappears, and then reappears. This happens because the word Hello is first written in white, then it is overwritten in the background color (assuming it is 0), and then rewritten in white. If you have color available, you can get some interesting effects by changing the color of a word on the screen.

Debugging and Testing

The distinction between *commands* and *functions* will become increasingly important as you work with *LinkWay* scripting. Keep in mind that a *function* produces a value. This value is then reported to a command. A *command* immediately takes some action. You cannot use a function by itself in a script. Every line of code in a script must begin with a command.

The commands **prompt** and **msg** can be excellent debugging tools. If you create a script that doesn't seem to work right, you can insert one of these commands in the script to see how much of the script works correctly. For example, inserting

```
msg "First three lines are OK"
```

could be inserted as the fourth line in a script. When you see this message, you know that the first three lines of the script are working correctly and that the problem is elsewhere.

New *LinkWay* Words in This Chapter

Commands
*drowcol *rownumber,columnnumber*
msg *expression, [expression]*
prompt *expression*
*text *color,x1,y1,x2,font,text-expression*

Functions
time
*itime

*Available only in *LinkWay Live!*

Project Ideas

Create a page that provides "time" information. If you are using *LinkWay Live!*, use **itime** to show how long it takes for some action to take place.

Experiment with **text.** What interesting effects can be achieved by using different colors to change the text? What effects can you produce by writing text on top of itself?

If you are using *LinkWay Live!*, experiment with **drowcol**. How wide and how tall is the screen? What values can you use in various modes? What happens if you use numbers near the bottom or the right side of the screen?

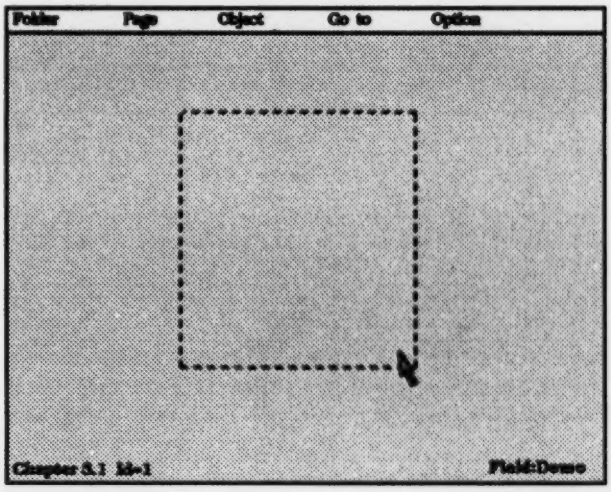
Spend some time experimenting with the ideas in this chapter. Do you understand the syntax of all of the new commands and functions discussed? Do you understand the difference between a command and a function? Do you understand the differences among **prompt, msg,** and **text?**

Chapter 3. Editing Text

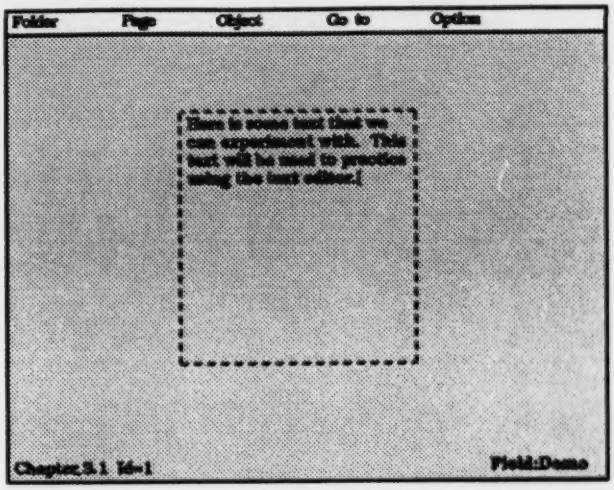
In this chapter you will learn about the text editor available in *LinkWay*. This editor makes it easier to modify the text in pop-up buttons, fields, and scripts.

You have worked with text in a number of ways in *LinkWay*. You have put text in fields. You have learned how to put text onto pop-up buttons. And you have learned to put text onto Script buttons. Perhaps you have had some difficulty modifying the text. In this chapter, you will learn easier ways to make changes in any of the editable text you create in *LinkWay*.

To begin, create a large field on a new page.



Name the new field Demo. Put some text in the field. Make sure it is several lines long so that you have enough text to experiment with.



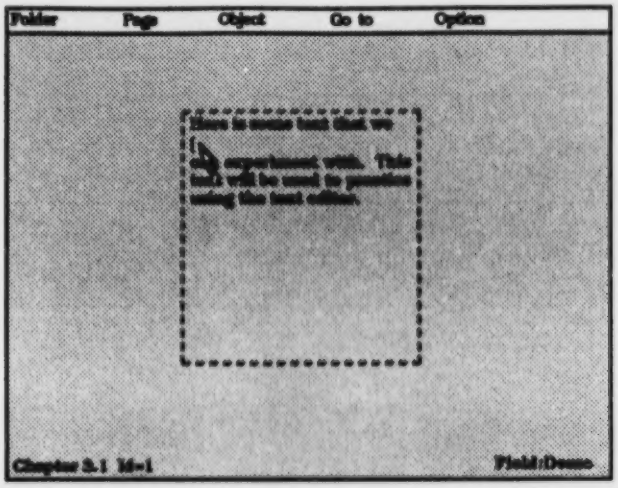
You have already learned that there are two modes for entering text: Insert mode and Replace mode. When using Insert mode, new characters move the existing text to the right. In Replace mode, new characters replace the existing text. When you see the [cursor, you are in Insert mode. When you see the | cursor, you are in Replace mode. Pressing the Insert (Ins) key switches between these two modes.

When you are in Insert mode, you can insert characters anywhere in your text. Simply place the cursor by using either the arrow keys or the mouse, and begin to type. Similarly, if you are in Replace mode, you can simply place the cursor and type over existing text.

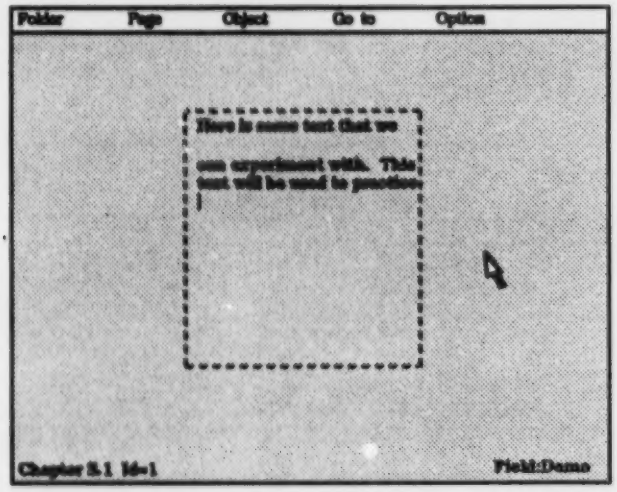
Deleting characters is also easy. Place the cursor in the text. To delete the *next* character, press the Delete key. To delete the *previous* character, press the Backspace key.

You can also insert a blank line into you text. Begin by putting your cursor at the beginning of the *second* line of text. You can use the arrow keys to move the [cursor, or you can use the mouse. To use the mouse to get the cursor to the left of the first character on the second line, click *on* the first character. Nothing happens when you try to click to the *left* of the first character.

Next, press either the F10 key or the Enter key. A blank line is inserted between the first and second line of text.

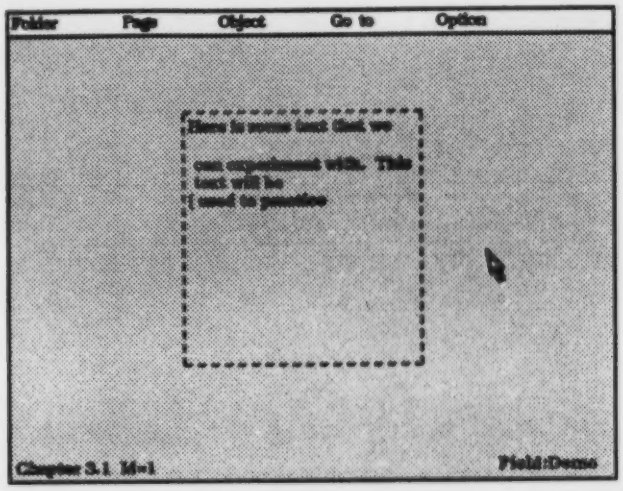


You can also delete an entire line. Move your cursor to the beginning of the last line of text. Press the F9 key. The line of text is deleted.



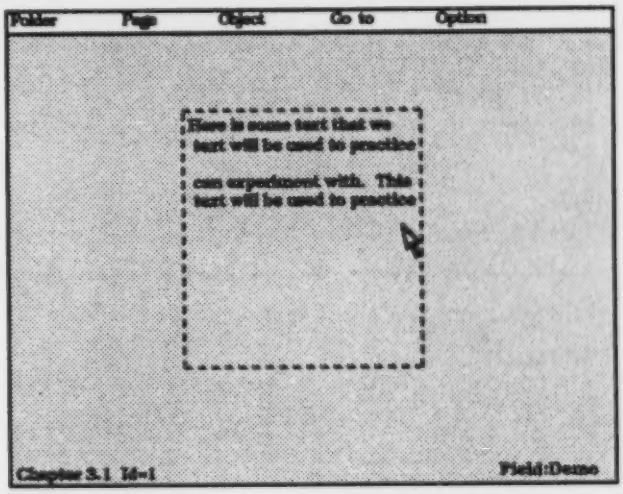
The deleted line is stored in a buffer that can hold up to 10 lines of text. Later you will learn how to retrieve these lines of text.

You can also easily rejoin lines that have become split. Again be sure that you have the [cursor. Move the cursor to the middle of the last line of text. Press either F10 or Enter. The line is split into two parts.

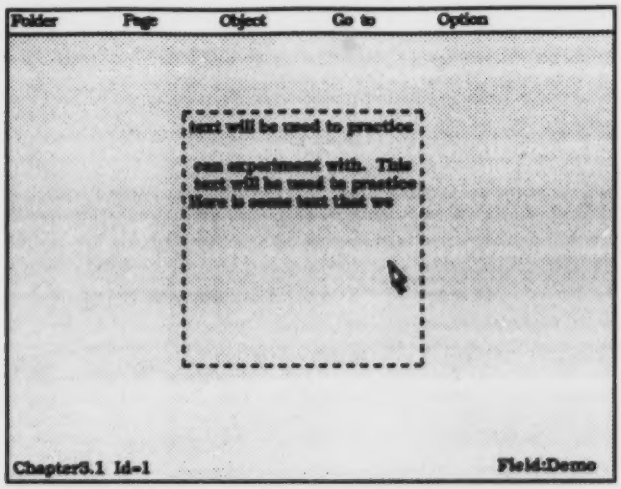


You can rejoin two lines by putting the cursor at the *end* of the upper line and pressing F9.

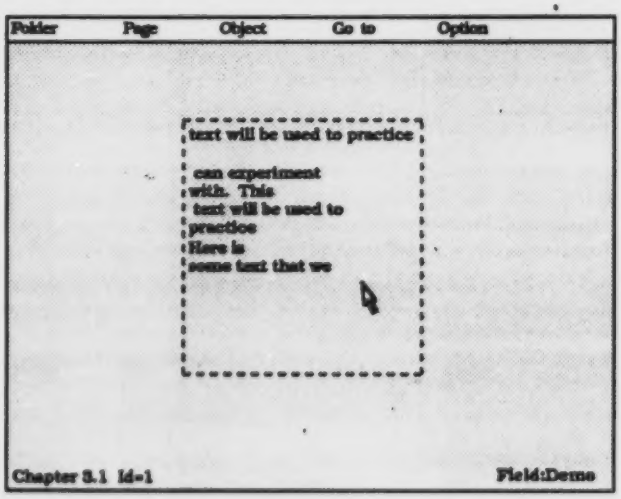
You can easily make copies of lines. Put the cursor at the beginning of the last line of text. Press F7. The cursor moves down a line. You have copied the line into the copy buffer. (This buffer can hold up to 10 lines of text.) Now move the cursor up to the blank line you created previously. Press Shift + F7. A copy of the last line appears, and the blank line moves down one line.



You can also move a line of text. Place the cursor at the beginning of the top line of text. Press the F9 key. The line disappears. Move the cursor to the end of the text in the field. Press Shift + F7. The line reappears at the end of the text.



Sometimes when you are working with blocks of text, they get shifted around in odd ways:



Move the cursor to the beginning of the block of text you want to put back together. Press F6. The text from the cursor to the next blank line is put back together.

Note that reflowing does not work with scripts. You must manually put lines of script back together with F9 if they become separated.

To stop editing, you can choose one of the following three options:

- Press either F2 or Esc
- Close the window

- Put the cursor on the last line of the field and press Enter. (Note that this is *not* the last line of text but rather the last line that may contain text in the field; i.e., the bottom right corner of the field.)

You should spend some time practicing the use of these various editing keys. It will save you a lot of time if you learn them well. At first you may want to make a copy of the following chart to use as a reminder. However, you will be constantly editing text and scripts as you continue your work in this book, so you will want to learn to use these editing keys without needing to refer to a reference sheet.

LinkWay Text-Editing Keys

Action	Keys to Use
Delete a character to the left	Backspace
Delete a character to the right	Delete or Del
Insert a blank line/Split a line	F10 or Enter
End Editing	F2, Esc, or Enter
Reflow text	F6
Copy a line	F7
Delete a line/Join two lines	F9
Move a line	F9 then Shift + F7
Switch between Insert and Replace Mode	Insert or Ins
Paste a line	Shift + F7
Insert a Character	Type character in Insert mode

Project Ideas

Spend some time practicing the use of the editing keys. Make some pop-up buttons. Make a field. Write some scripts. Use the editing keys to modify all of these. Be sure you feel comfortable with all of the actions described in this chapter.

Examine the editing keys carefully. Look for patterns. For example, F9 is used in a number of places. What does F9 *really* do? (Note, for example, that F9 simply deletes and puts whatever it deletes in the buffer. When you use F9 to join two lines, it merely deletes the ending characters that mark the end of the first line.)

Chapter 4. Working With Objects

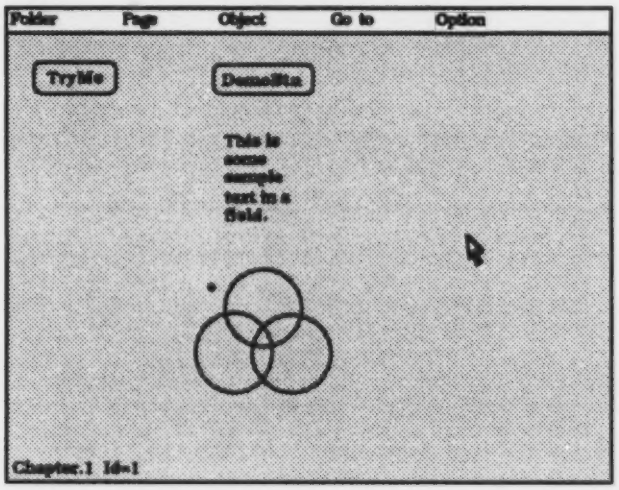
In this chapter you will learn about ways to work with objects from scripts. In particular, you will learn how to use script buttons to modify and change the objects on a page.

In your initial work with *LinkWay*, you learned to work with a number of objects on the page: fields, buttons, and pictures. Each of these objects has a number of characteristics, such as size, shape, and position. In this and subsequent chapters, you will learn that you can modify these characteristics by using script buttons.

To get started in this chapter, create four objects:

- a Go button called DemoBtn
- a script button called TryMe
- a field named DemoFld
- a picture called DemoPic

The picture can be either something you create using *LinkWay Paint* or a piece of art from another source. Place the objects on the page as shown in the following illustration. Don't put any script on the TryMe button for now. Put some text in the field. You may want to make the information in the field and the picture more meaningful than that shown. The exact contents of the page are not important at this point.



BEST COPY AVAILABLE

Modifying Objects

You can make changes to individual objects on your page from a script. First, an object needs to be selected with the **object** command and then characteristics of the object can be modified with the **change** command. The syntax of **object** is:

```
object objectname
```

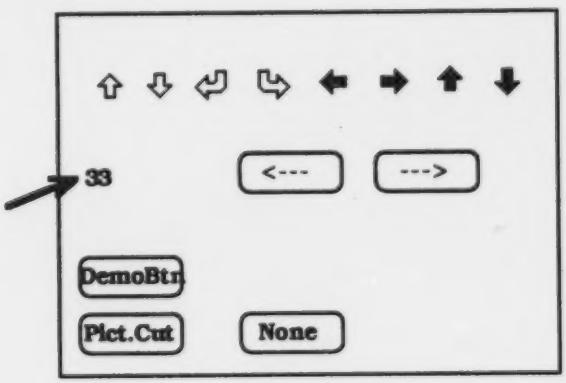
The syntax of **change** is:

```
change attribute
```

Modify the script of the TryMe button as follows. Remember to use the editing keys you learned in the last chapter.

```
object "DemoBtn";
change 49;
wait 0.5;
change 0;
wait 0.5;
change 49;
```

When **change** is used with buttons, it changes the icon on the button. If you click on the TryMe button, the button DemoBtn changes to show a small telephone. The phone disappears and reappears. The number after **change** is the icon number. If you look carefully at the Button Icon window, you will see a number at the left side of the window. This is the number of the left-most icon in the window. The icons are numbered sequentially to the right. Thus, in the following diagram, the black arrow pointing down is icon number 40.



In the standard Button Icon window, 49 is a small telephone. In addition to the numbers you can find in the Button Icon window, there are some other conventions used by **change** with buttons:

- Adding 128 to an icon number makes it visible but not clickable.
- Setting the icon to 0 makes the icon invisible.
- Icon number 128 is invisible and not clickable.
- Icon 98 is a bit cut. (See Chapter 19 for more information on picture cut buttons.)
- Icon 99 is a sign. (Recall that a sign is the word used to describe a button with the name inside of a rectangle.)

Notice that by using these various icon numbers with **change** you can do such things as make buttons temporarily invisible, make buttons that are visible that the user can't activate, and select predefined graphics for use on a button. This use of **change** can be valuable if you want to temporarily disable a button on a page or create a special effect by modifying the button.

In order to refer to objects in scripts, the name of the object must begin with a letter and include only letters, digits, and the underscore (_) character. Do not use spaces in object names that will be used in scripts.

If the name specified after **object** is a field, the number after **change** is the color of the field. If you put a 0 after **change**, the field disappears. Try this script on the TryMe button:

```
object "DemoFld";
change 1;
wait 1;
change 0;
wait 1;
change 1;
```

If you are using color, try changing the numbers after **change**. (See your *LinkWay* manual for the color choices you can use.)

For pictures, the information after **change** is the file name of the picture to be displayed. Thus, you might try something like this:

```
object "DemoPic";
change "Pic1";
wait 2;
change "Pic2";
```

The screen switches from a picture called Pic1 to a picture called Pic2. Be careful! The picture you display must be positioned so that it appears in the window you created for your picture object DemoPic. Using **change** in this manner lets you display a slide show with the click of a button.

Next, edit the TryMe button again and replace the previous script with this one:

```
bstyle 1
go seq
```

The command **bstyle,** *which changes the style of buttons, is only available in* LinkWay Live! *Only one style of button can be used in* LinkWay.

Click on the TryMe button. You see that the style of the button has changed from the default round rectangle used for buttons when you start *LinkWay Live!* to a plain box. The **bstyle** command sets the style of all the buttons in the folder as they are redisplayed. There are three button styles:

- 0 shadow box
- 1 plain box
- 2 rounded corner box (default button style)

The syntax of **bstyle** is:

```
bstyle number
```

The command

```
go seq
```

causes the same page to be redisplayed again, and thus causes the buttons to redraw in the new style. You will learn more about **seq** in Chapter 15.

If you have a computer that redraws pages quite rapidly, you can use this technique to draw attention to buttons. For example, you might write this script:

```
bstyle 2;
wait 1;
go seq;
bstyle 1;
wait 1;
go seq;
bstyle 2;
wait 1;
go seq;
```

The buttons change from a round rectangle to a rectangular box and back to a round rectangle. Note that **bstyle** changes all of the buttons to the new shape *as they are redisplayed.* It does not change the shape of the button with a script on it.

Cutting, Pasting, and Removing Objects

There are three commands that let you manipulate objects in your folder. These commands are **ocut, opaste,** and **odelete**. These let you cut an object and place it in a file on disk. **Opaste** takes an object from a file and puts it on the page. **Odelete** removes an object from the page. The

object command can be used to select the object. Enter this script on the TryMe button:

```
object "DemoPic";
odelete;
```

When you click on TryMe, the picture object is removed from the page. The syntax of these commands is:

```
odelete
ocut [filename]
opaste [filename]
```

The name after both **ocut** and **opaste** is assumed to be Scrap unless another file name is entered.

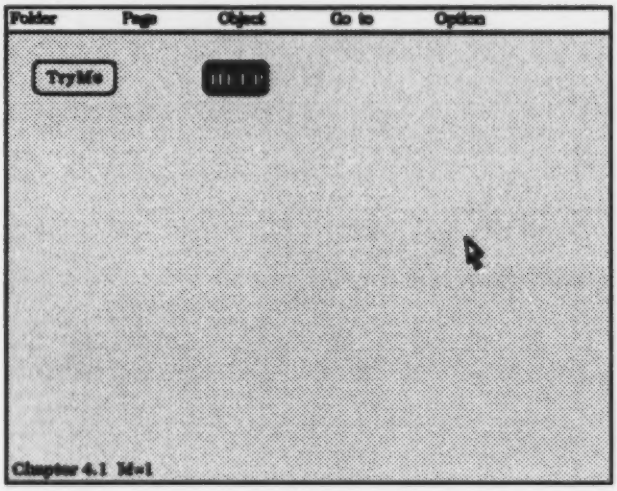
You might use **ocut** to remove an object from a page. You could then use **opaste** to paste a different object on the page. At a later time you could restore the first object with **opaste.** Objects removed from the page with **ocut** reappear in the same place when they are pasted with **opaste.**

Moving Objects

The commands **move** and **wmove** are used in scripts to move objects on the page.

*The command **wmove** is available only in* LinkWay Live!

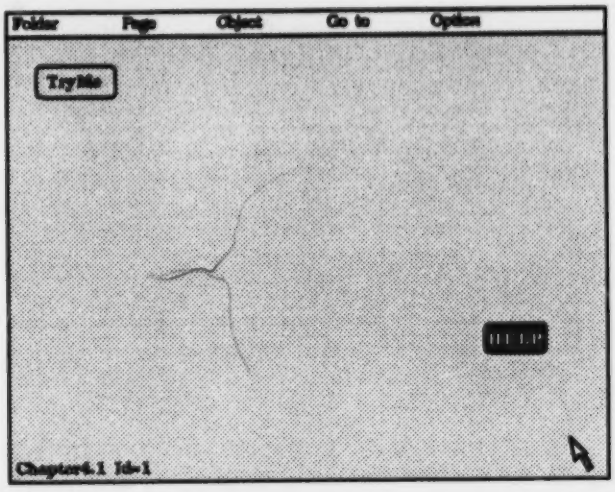
For now, remove the field from the card and set the icon of button DemoBtn to 7, the small "HELP."



Put the following script on the TryMe button:

```
object "DemoBtn";
move 30,20,35,22;
```

Click on TryMe. The button moves to the right and down. If you click on the button DemoBtn, you will most likely see that its bounding box has changed shape. The **move** command puts the button in a new location *and* establishes the shape of the bounding box.



The exact location of the button on *your* screen may be different than that shown in the previous diagram. The number of rows and columns available on the screen depends on the mode you are using. **Move** uses character positions. The range of rows may be 0-24 or 0-29, while the range of columns may be 0-39, 0-44, or 0-79. See your *LinkWay* documentation under "move" for details.

The syntax of **move** is:

```
move left, top, right, bottom
```

Left and right are the left-most and right-most columns of the bounding box; top and bottom are the top-most and bottom-most rows of the bounding box.

If you are using *LinkWay Live!*, there is a command similar to **move** called **wmove**. **Wmove** uses "world" coordinates. Its syntax is much like that of **move**:

```
wmove left, top, right, bottom
```

However, when using world coordinates, the range across the screen is 0-159 while the vertical range is either 0-99 or 0-119 depending on the mode you are using. Again, see your *LinkWay* documentation for details for the mode you are using.

Move and **wmove** can be used to animate objects. For example, put this script on the TryMe button:

```
object "DemoBtn";
move 10, 10, 15, 12;
wait 0.5;
move 12, 12, 17, 14;
wait 0.5;
move 14, 14, 19, 16;
wait 0.5;
move 16, 16, 21, 18;
```

You see the button move down the page and to the right. You could also use this technique to "unveil" a picture. Suppose you have a picture object named DemoPic. It needs to be a fairly large picture. Enter code like this on the button TryMe:

```
object "DemoPic";
wmove 75, 45, 85, 55;
wait 1;
wmove 65, 35, 95, 65;
wait 1;
wmove 55, 25, 105, 75;
wait 1;
wmove 45, 15, 115, 85;
```

The bounding box for the picture gets larger four different times. Each time it gets larger, more of the picture appears.

Debugging and Testing

As your scripts get longer, more problems arise. Thus, you need to further develop your debugging. Are you remembering to use **prompt** or **msg** to help tell where problems are occurring in scripts? Are you remembering the semicolons at the end of each statement? While some statements will work correctly without the semicolons, others will not. Check your scripts carefully.

The **object** command adds another level of problems. You must be careful to spell the name of the object correctly. A misspelled object name will result in the script not working. Always be careful not to confuse the name of a picture *file* and the name of the picture *object* on the page.

The graphics mode you are using in *LinkWay* is important to your debugging. Different graphics modes mean that different coordinate ranges are available to you. It is a good idea to check the graphics mode on the **Option** menu each time you begin working with a new folder.

New *LinkWay* Words in This Chapter

Commands
*bstyle *number*
change *attribute*
move *left, top, right, bottom*
object *objectname*
ocut *[filename]*
odelete
opaste *[filename]*
*wmove *left, top, right, bottom*

*only available in *LinkWay Live!*

Project Ideas

Explore the styles of buttons and attributes of buttons, fields, and pictures. Practice writing scripts that use **change** to modify the objects.

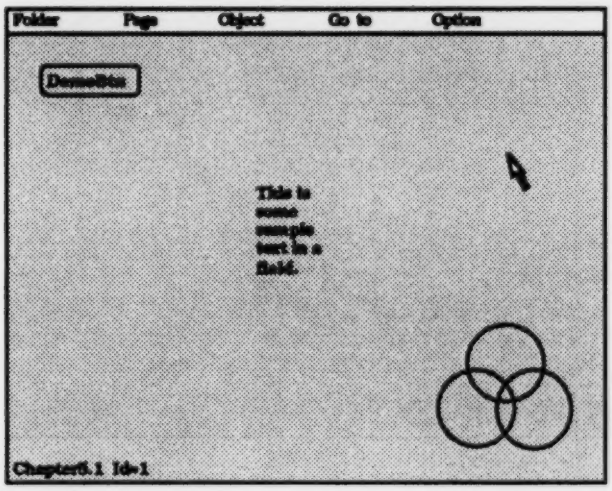
Create a page with a number of objects on it. Use **object** to select objects. Remove them and paste them again using **ocut** and **opaste**. Then try moving objects and pasting new objects in the place of old ones. Use file names other than Scrap to store the objects.

Spend some time exploring **move** and/or **wmove**. What is the range of coordinates you can use in **move**? What is the range of coordinates you can use in **wmove**?

Chapter 5. Shortcuts and Tools

In this chapter you will learn to use the keyboard to speed up the editing of the objects on your page. Note that many of the keystrokes are available only in *LinkWay Live!* In addition, you will learn about the availability of a number of useful tools that are available in both *LinkWay* and *LinkWay Live!*

To begin your work in this chapter, create a page containing three objects: a button, a field, and a picture. You may want to use the experimental page from the previous chapter. Name your button DemoBtn, your field DemoFld, and your picture DemoPic. Place them on the page as shown here:



Keyboard Shortcuts

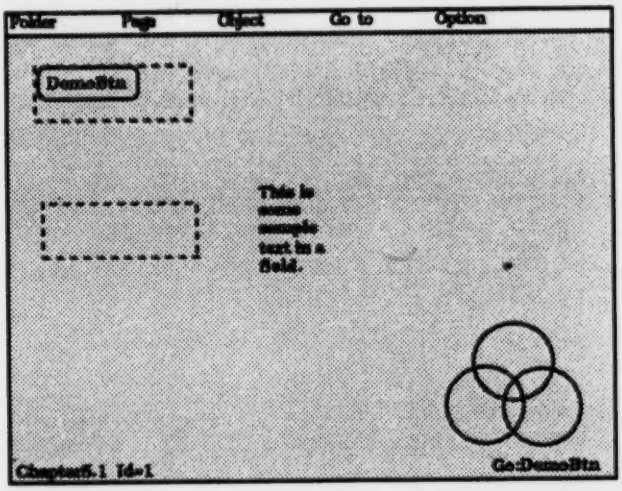
When your page is set up as shown, press the Tab key several times. As you press the key you see the bounding boxes for the objects on the page appear. If you created the button first, then the field, and then the picture, you first see the bounding box of the button. Then you see the bounding box of the field. Then you see the bounding box of the picture, and then you see no bounding boxes. Repeatedly pressing the Tab key cycles through the objects in the order they were created.

Now, try selecting the field with the mouse. Press the Tab key. The bounding box of the *next* object created—probably the picture—is highlighted. The Tab key does not always begin with the first object

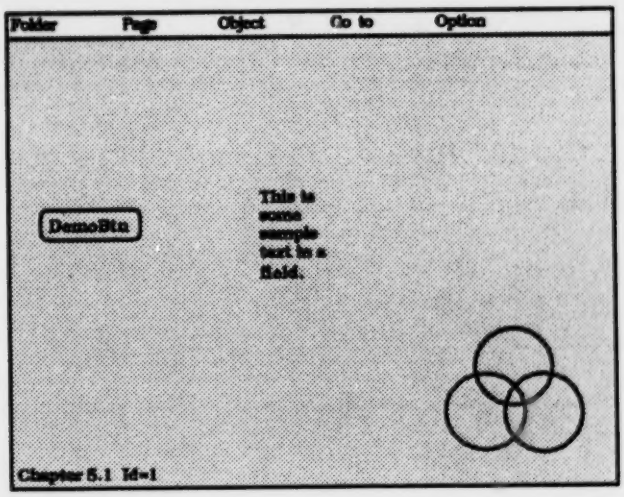
created. If the last action you performed before pressing the Tab key was to select an object, the Tab key will begin at that point in the sequence.

The keys described in the rest of this section work only in LinkWay Live! *If you are using* LinkWay, *the Tab key is the only key shortcut for editing objects.*

Next, select the button on your page. While the button is selected—and the bounding box is visible—press the M key. A second bounding box appears at the location of the mouse pointer. Move this bounding box to the new location for the button.

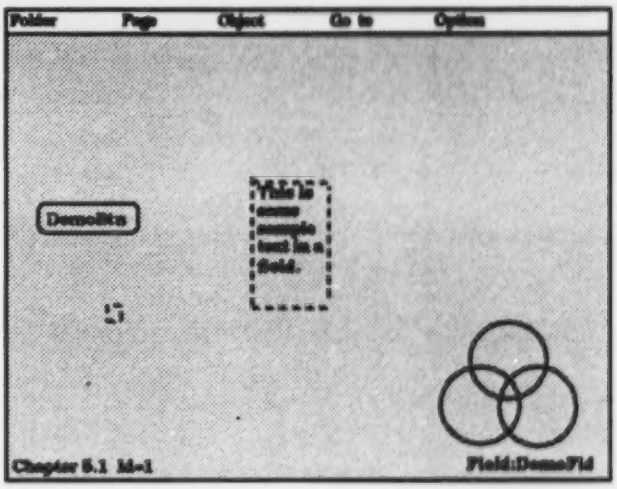


Click the mouse button. The button reappears at the new location.

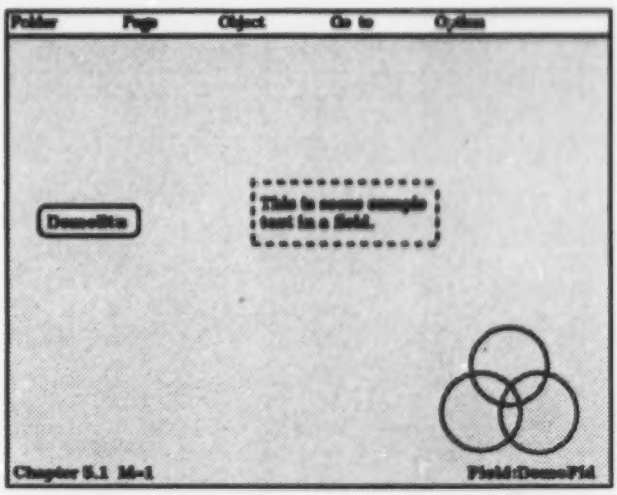


You have moved the button without using the **Object** menu. Pressing M when an object is selected is the same as pulling down the **Object** menu and clicking on **Move.**

You can also do a **Move + Size** by using key shortcuts. Select the field. A small square appears at the location of the mouse pointer.



Use the mouse to position and resize the field. When you are satisfied with the new location, click the mouse.



Similarly, you can use key shortcuts in *LinkWay Live!* to edit any object by using keystrokes. Select the object. Press the E key. The first of the sequence of windows for that object appears. Use the mouse and keyboard as usual to edit the object.

BEST COPY AVAILABLE

You can also delete objects without using the **Object** menu. Select the object and press the D key. Be careful when using this key. You can easily delete objects you don't want to remove. Pay careful attention to which bounding box is visible and to the information in the lower right corner of the page.

You can use the B key to change the color used to display dotted object outlines. First select the object with the Tab key and then press B. Repeatedly pressing B causes different outline colors to appear. Note that this key only works in MCGA-256, VGA-8, and XGA modes.

Finally, you can use Alt + Q to leave *LinkWay*. The current folder is automatically saved when you use this key combination.

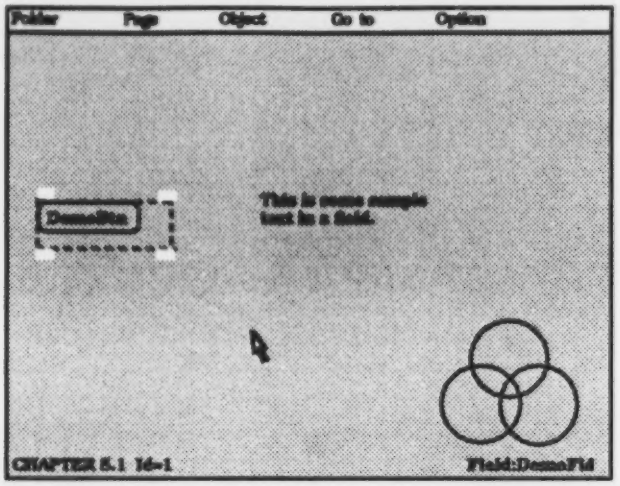
Using Expert Mode

When you start *LinkWay Live!* there is a button on the opening page labeled Profile. By using this button you can change the configuration of *LinkWay Live!*

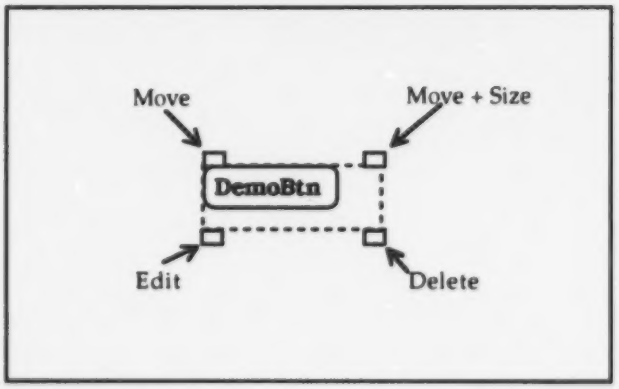
> *If you are using* LinkWay *rather than* LinkWay Live!*, this option is not available.*

Return to the Main folder, and click on the Profile button. A list of settings appears on the screen. At the bottom is a place to choose between a System-wide profile and a Private directory profile. If you are using a public version of *LinkWay Live!*, be sure you click on the dot next to Private directory profile. The last option in the list of items that can be changed is User type. Click on the setting to the right of User type, most likely "novice." A window appears. Click on Expert. Save your profile and return to the folder you were working with.

Now select an object. A bounding box appears with small rectangles at each corner. The rectangles are called *bounding box targets.*



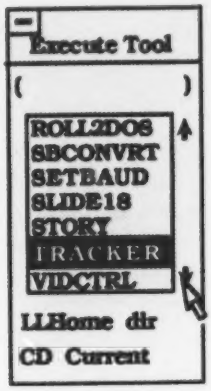
Clicking on each of these bounding box targets activates a different object-editing function. The function of each target is as follows:



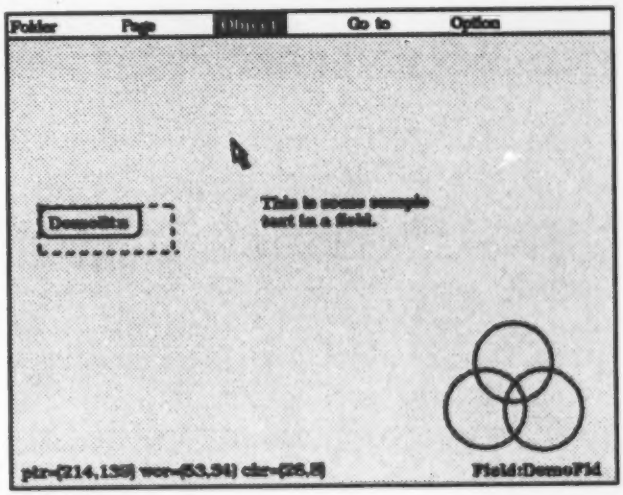
Note that because this feature is available only in *LinkWay Live!* and because some readers may prefer to work in the Novice mode, bounding boxes in the rest of the book will be shown *without* the bounding box targets.

Using Tools

Both *LinkWay* and *LinkWay Live!* come with a number of tools. *LinkWay Paint* is one of those useful tools. To explore how to use these tools, go to the **Object** menu and select **Tools**. The Execute Tool window appears.



Click on the down arrow until you see Tracker. Select it from the list. A window appears that explains the function of Tracker. Click again to remove the window. Three sets of coordinates appear in the Status Line.



These numbers represent the coordinates in pixels (pix), in world coordinates (wor), and in character columns and rows (chr). Move the mouse. You see the numbers change. Move the mouse to the upper left corner of your page. Move the mouse to the lower right corner of the page. You might want to write down the range of values possible on your computer for each of these three sets of coordinates. When you are finished exploring, click the mouse again.

As you can see, Tracker is a valuable tool to help you position objects on the page. Perhaps it would be helpful to have Tracker available without going to the **Object** menu each time. You can accomplish this quite easily.

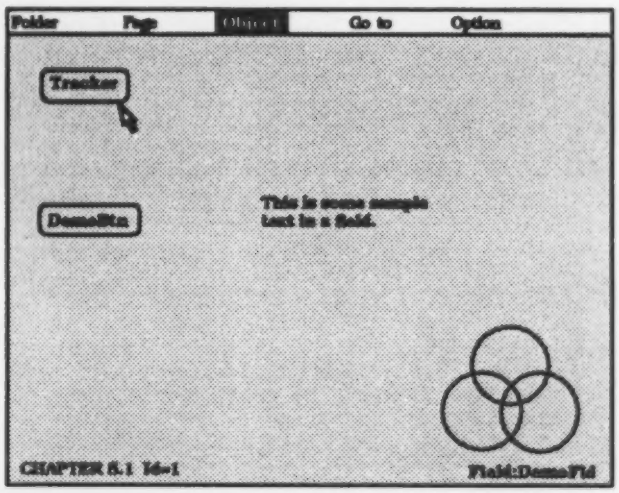
To get a Tracker button on your page, first select **Paste** from the **Object** menu. In the Paste Object window, click on CD LLHome.

Click on CD LinkWay Path if you are using LinkWay *rather than* LinkWay Live!

You see a list of tools. Select Tracker.



Close the Paste Object window. A bounding box appears. Click on your current page. A Tracker button appears.



You can now use the Tracker button any time you wish.

There are several other tools available with your version of *LinkWay*. Check your manual for a description of the tools you can use. Explore how some of these tools work. You might like to try Story. It allows you to "flip through" the pages in your folder. You also might like to try DragObj. This button lets you drag objects around the page by using the mouse.

Debugging and Testing

In this chapter you learned a number of shortcuts for speeding up your work with *LinkWay*. It is wise to take the time to master each of the techniques described. The small amount of time invested now will save you much effort later on.

Tools like Tracker and DragObj can be very helpful when you are designing a folder or debugging a problem on a page. You might consider developing a page of useful tools that you could paste into each folder you are working on. This page could then be deleted when you are finished with the folder. Such a folder might have useful buttons like Tracker and DragObj. It also might have some objects configured in the ways that you often use them. For example, you might have a "next page" and a "previous page" button. Each of these buttons could have the arrows on them that you prefer to use. Similarly, you could have a picture object defined. When you copy this object to a new page, you only need to change the picture you want to use, or perhaps the size. Predefined objects such as this serve as templates for creating new objects on new pages. This technique can save a lot of time.

Think of other ways you can streamline your work with *LinkWay*. When you first started to work with *LinkWay*, you needed to use all of the menus and to create objects from scratch. As you become more expert, you will want to skip some of those early steps. What other techniques could you use to speed up your work?

Key Shortcuts

Action	Key	Expert mode
Selects next object	Tab	
Delete selected object	D	Lower right target
Edit selected object	E	Lower left target
Move and size selected object	S	Upper right target
Move selected object	M	Upper left target
Change the bounding box color	B	
Exit *LinkWay*	Alt + Q	

Project Ideas

If you are using *LinkWay Live!*, practice using the key shortcuts described at the beginning of this chapter. Be sure you are comfortable working with objects by using the Tab key and the shortcut keys. Explore

the use of Expert Mode as well. Decide which method of editing you like best and be sure you have mastered it.

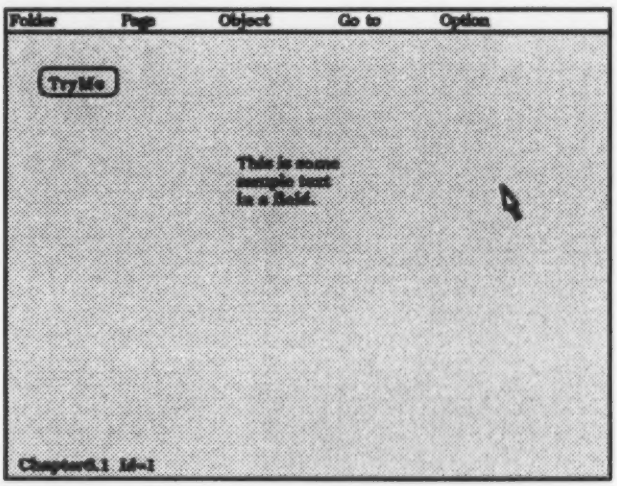
Spend some time exploring the tools that come with your version of *LinkWay*. Make note of those you think you might find useful in the future.

Create your own Tools page that contains your favorite tool buttons and your favorite configurations of buttons, fields, and pictures.

Chapter 6. Working With Fields

In this chapter, you will learn some ways to put text into fields from scripts. You'll also learn another way to select objects. Finally, you will learn to work with different fonts in fields.

To begin your work in this chapter, create a field called DemoFld and a Script button called TryMe.



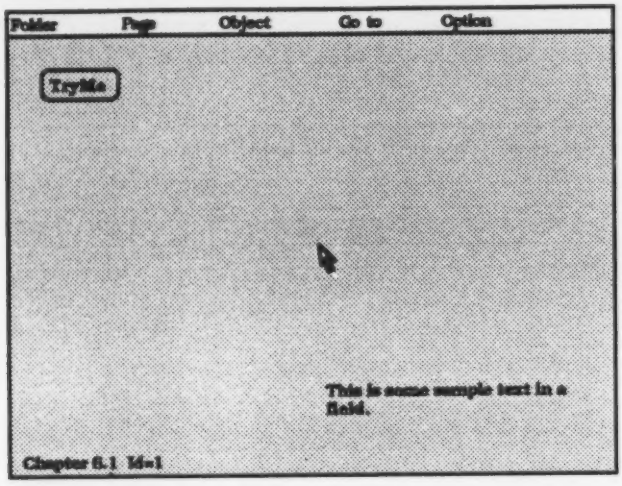
Selecting Objects

You have learned to select objects from scripts by using the **object** command. There is another way to select an object. Put the following script on the TryMe button:

```
msg "Click on an object";
mouse;
move 35,18,39,20;
```

When you click on the TryMe button, you see a window with the words "Click on an object." Click on the field. It moves to the right toward the bottom part of the page.

BEST COPY AVAILABLE



Note that the position of the field will vary depending on the mode you are working in.

The **mouse** command causes *LinkWay* to pause until the user clicks the mouse. The syntax of **mouse** is:

```
mouse [timeout]
```

If there is no number given after **mouse**, *LinkWay* waits for the next mouse click. In the previous script, *LinkWay* then runs the **move** statement for the selected object. If no object is selected, a syntax error appears.

The *timeout* input to **mouse** can be used to wait a specified time for a mouse click. Thus,

```
mouse 10
```

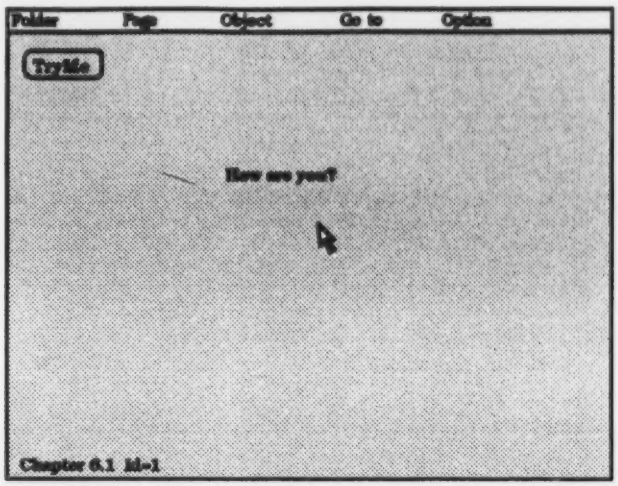
waits for 10 seconds for a mouse click. If there is no mouse click, the next statement in the script is run.

Putting Text in Fields

There are two commands, **set** and **flowset**, that allow you to put text into fields from scripts. Move DemoFld back to the middle of the page and modify the script on TryMe as follows:

```
msg "Click on an object";
mouse;
set object = "How are you?";
```

When you click on the button, you are asked to click on an object. Click on the field. The words "How are you?" appear in the field.



In this command, **object** is a *function* that reports to *LinkWay* the name of the currently selected object. If a different object were selected with the mouse, **object** would report a different value. The syntax of the *function* **object** is:

```
object
```

Of course, you can get the same result by using the **object** *command:*

```
object "DemoFld";
set object = "How are you?";
```

The **set** command is used to pass a value to the **object** function. The syntax of **set** is:

```
set object|objectname = text-expression
```

The "object|*objectname*" after **set** means that you either can use the word "object"—representing the currently selected object—after **set**, or you can use the specific name of an object. That is,

```
object "DemoFld";
set object = "How are you?";
```

is equivalent to typing

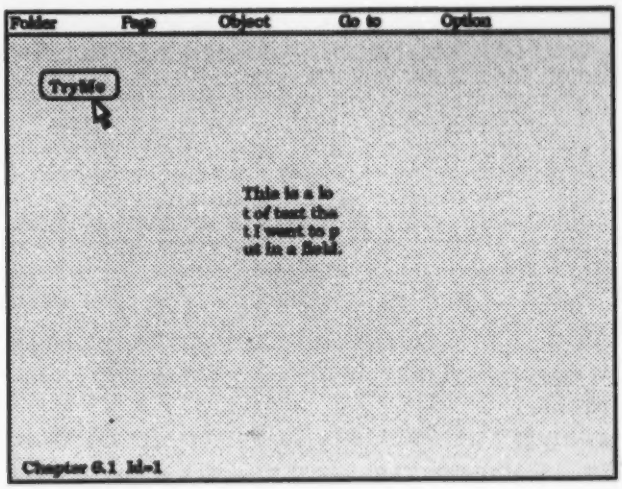
```
set DemoFld = "How are you?";
```

There are many other ways in which **set** can be used. They will be discussed later in the book (see Chapter 7). Note that this process of using **set** to assign text may be used with pop-up buttons.

Suppose you want to put a lot of text in a field. You might have a command such as this:

```
set DemoFld = "This is a lot of text that I want to
   put in a field."
```

Your text may look like this:



To correct this problem, you can use **flowset**:

```
flowset DemoFld = "This is a lot of text that I want
   to put into a field."
```

The text now breaks between words instead of in the middle of words. The syntax of **flowset** is much like **set**:

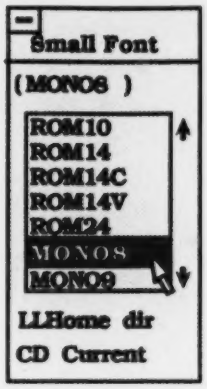
```
flowset objectname=textexpression
```

The named object must be either a field or text pop-up button.

Working With Fonts

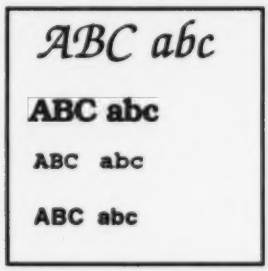
So far in your work with *LinkWay*, all of the text you have created has been in the same font. Although you can have only one font in any field, you can change two of the four fonts available when you create a field. The top font in the list of available fonts is referred to as the Large Font. The third font is referred to as the Small Font.

Go to the **Option** menu and select **Fonts**. The Small Font window appears.



Select one of the Mono fonts. Next, the Large Font window appears. Select Script if it is available.

Next, edit the field DemoFld. You see something like this:



The first font you selected—the Small Font— appears as the third font in the list. The second font you selected—the Large Font—appears at the top of the list. Choose one of the new fonts and type in your field. Note that the default fonts are Hel and Rom.

Debugging and Testing

Are you having increasing difficulty finding the errors in your scripts? Those mistakes are called *bugs*. The process of correcting the errors is called *debugging*. The more complex your scripts become, the harder it can be to find your bugs. Remember to pay careful attention to spelling and punctuation. Be sure to put semicolons after each statement. Pay attention to when quotations marks are needed, for example, surrounding strings or the name of objects when using the **object** command.

All programmers at one time or another have difficulty debugging their scripts. When you are having difficulty with your scripts, it is good to take a break. Often, just getting away from the computer for a while makes things work better when you return. Don't do the same thing over and over and over. Ask for help. Read your manual. Take a break and try

again. Remember that even professional programmers spend a *large* percentage of their time debugging their work.

The word *object* is used in a variety of ways in *LinkWay*. This can be quite confusing. **Object** is a command that lets you select an object in a script by putting the name of the object to select after the word **object**. **Object** also is a function that holds and reports the name of the currently selected object. Finally, the word *object* is used in the **set** command to get the name of an object to "pass to" the **object** function. Thus, when **object** is the first word of a *LinkWay* statement, it is a command. If the word *object* appears after **set**, it is neither a command nor a function. If **object** appears elsewhere in a statement, it is a function.

New *LinkWay* Words in This Chapter

Commands
`flowset object|objectname = text-expression`
`mouse [timeout]`
`set object|objectname = strong`

Functions
`object`

Project Ideas

Create a page that has several fields. Have buttons for putting information into the fields. Use **mouse** to let the user select which fields to use. Be sure to include instructions for the user.

Spend some time looking at the different fonts available with your version of *LinkWay*. Create some sample fields showing the different fonts. You might want to save a page with samples of all of the fonts on it for future reference.

There are several ways—**mouse**, **set object**, and **set** *objectname*—to select an object. Explore these three methods of selecting objects in particular fields. When would you use each method?

Chapter 7. Overview of *LinkWay* Grammar

This chapter focuses on the structure of the *LinkWay* scripting language. In particular, you will review what you have learned about commands and functions. In addition, you will begin to explore some other features of the scripting language.

Programming languages, like "natural" languages (English, French, Chinese, Russian, and so forth), have rules of grammar. Natural languages can be very complex, taking many years for a nonnative speaker to learn. They have many words in their vocabularies and the uses of these words tend to be filled with exceptions and special cases. Programming languages are much simpler to learn than their natural-language counterparts. However, even among programming languages there is great variation. For example, the simplest versions of the BASIC programming language have only 15 to 20 commands and functions to learn, while modern versions of the Logo programming language often have more than 200 special words. On the other hand, the syntax rules of the Logo programming language can be described in a few paragraphs, while the rules for languages such as Pascal can take pages of diagrams. In addition, some languages are virtually without "special cases." That is, each and every statement in the language exactly follows the same simple set of rules of the language. Other programming languages have a number of special cases that often exist to make working with the language easier for the computer. However, the programmer must remember these special cases in order to write correct programs.

The *LinkWay* scripting language is neither the simplest nor the most complex of the programming languages. There are some special cases, but in general it follows its own grammatical rules quite well. While it has a fairly large number of special words, they are very English-like and thus relatively easy to remember.

Commands and Functions

Previously in this book, you learned about commands and functions in *LinkWay* scripting. Recall that commands perform an action immediately, while functions merely generate values that are used by *LinkWay* commands.

Your LinkWay *documentation uses the word "statement" for* LinkWay *commands, but it* also *uses the word "statement" for a line of* LinkWay *code. To avoid this confusion, this book uses the word "command" to refer*

*to the special words (**beep, wait, msg,** and so forth) and uses the word "statement" to refer to a line of code containing a command and its inputs.*

Consider the **prompt** command. The following *LinkWay* statement causes "Hello" to appear in the Status line:

```
prompt "Hello";
```

However, if you put

```
time
```

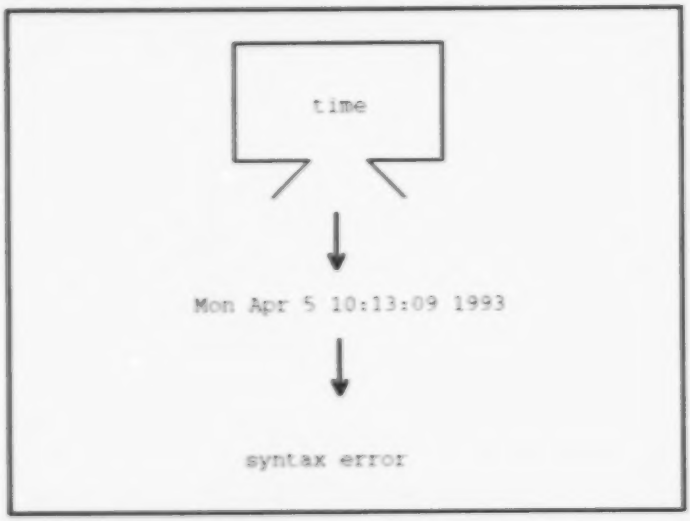
by itself on a line in a Script button, you get a syntax error message. The **time** function produces a value representing the current time. When you put this function in a script with no command preceding it, *LinkWay* has no place to put the value produced by **time;** therefore, an error message is produced.

Of course you can use a command to tell *LinkWay* what to do with the value produced by the function:

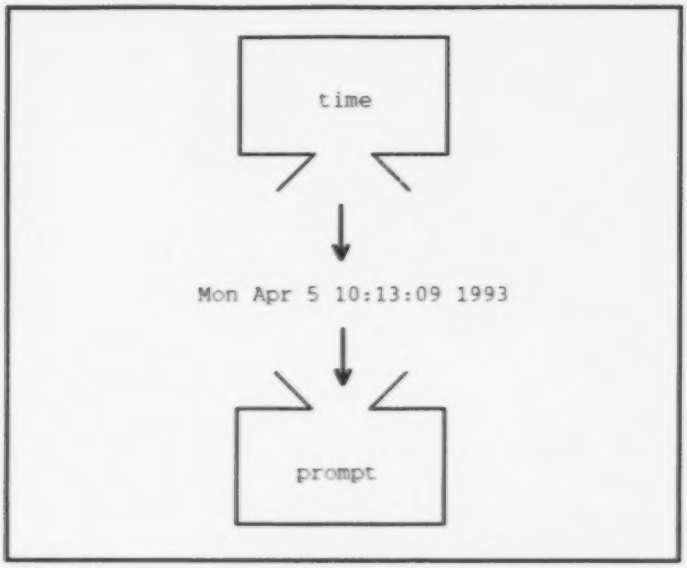
```
prompt time
```

In this case, the function **time** produces a value, and that value is "given to" the command **prompt**, which then puts it in the Status line.

You might think of functions as "funnels." If they have inputs, they accept values from their inputs and produce a result. If there is no command to "catch" the result, an error message results:



However, if the **prompt** command is present, the result of the function is used by **prompt** to display the value on the Status line:



Take a few moments to think about the difference between commands and functions.

Syntax Descriptions

In this book, syntax descriptions are given for each new command and function you learn. Take a few minutes to review the form of these syntax descriptions. Here is a more complete description of the **set** command, which you saw in Chapter 6. Recall that commands are shown in a single-line box.

```
set buttonname = expression
set fieldname = expression
set object = expression
set variable = expression
```

Notice that the first input to **set** can be a button name, a field name, the word "object," or a variable. (You will learn more about variables in Chapter 23).

Recall that your LinkWay *documentation refers to the inputs to commands and functions as "operands."*

Thus, you could write

```
set DemoBtn = "How are you?";
```

to assign the string "How are you?" to a Script or a Text Pop-up button.

You can also assign text to Link buttons, Reference buttons (Chapter 8), and Media objects (Appendix 6).

Similarly, if you write

```
set DemoFld = "How are you?";
```

the string on the right is placed in a field.

Recall that **object** can be either a command or a function. As a command, its syntax is:

```
object objectname
```

Here **object** must have one input—the name of the object being selected. However, as a function, its syntax is:

```
object
```

There is no input. The *function* **object** automatically holds the name of the current object.

Recall the syntax of the command **msg**:

```
msg expression [,expression]
```

Msg can have either one or two inputs. When there are square brackets around an input in a syntax diagram, that input is optional.

To summarize, in syntax diagrams in this book:

- Commands are shown in boxes with a single-line outline. Functions are shown in boxes with double-line outlines.
- Anything in square brackets–[ ]–does not have to be included, e.g., the second "string" in **msg**.
- Words in italics are called *inputs*—or *operands* in your *LinkWay* documentation. Particular values must be substituted for these inputs.

Sometimes the input to a command or function is a constant. A constant can be a quoted string, for example, the following input to **prompt**:

```
prompt "Good morning!"
```

The other type of constant is numeric. Numeric constants include digits, the - and + signs, the decimal point, and commas.

Often in syntax diagrams you see *expression* as the input. An expression can be a variable (Chapter 23), an object, a constant, a function, or a combination of these joined by *operators*. Operators are such things as

+ and -, ( ), and < or =. (See Chapter 22 for a more complete description of operators.) Later in this book, you will see such statements as

```
set newtime = itime + 10;
```

in which "itime + 10" is an expression made up of a function, a plus sign, and a number. Or you might see

```
move oleft + 10, otop, oright + 10, obot;
```

in which elements such as "oleft + 10" are expressions that provide input to the **move** command.

Unless you are doing very technical work, if you simply apply the rules you learned in math classes you will not have any problems.

Take a few minutes to review the commands and functions that you have learned so far. Go back and look at the summaries at the end of the previous chapters.

Using English-like words such as *input* in syntax descriptions makes them fairly easy to read and understand. However, computer languages are very precise so that "literal-minded" computers can understand them. On the other hand, natural languages like English do not require a high degree of precision to be understood. When a computer language is described in an English-like manner, it is hard to make the precise meaning of the computer language clear. There are a number of techniques that computer scientists use to specify computer languages exactly. One such technique is called Bakus-Naur notation. Bakus-Naur notation uses special symbols, such as { }, < >, [ ], and |, to completely describe a command or function. In the description for **set**, we borrowed the | symbol, which stands for "or":

```
set object|objectname = string
```

A person who understands notation like Bakus-Naur can easily determine exactly what the "legal" forms of a command or function are. However, this notation is not very readable for a novice. Thus, this book sacrifices precision for readability in its syntax descriptions.

Objects

Almost all programming languages have commands and functions. The tasks that you can accomplish using the language revolve around creating statements by using the commands and functions available in the language. However, *LinkWay* is somewhat different from languages like Pascal and BASIC. *LinkWay* is related to a category of programming languages called "object-oriented" languages. These languages rely on the concept of an *object*, as does *LinkWay*. In a true object-oriented programming environment, programmers can create their own objects.

However, the *LinkWay* programmer is limited to a variety of forms of the objects provided by *LinkWay*: buttons, fields, and pictures. Thus, *LinkWay* can be referred to as an "object-like" programming environment.

The strength of objects is that they carry their properties with them. For example, suppose you create a button that is of type Script and then you have it display a right arrow and put a script on it that reads

```
beep;
beep;
```

If you then cut and paste the button, the new button will have all of the same properties. It will have the same icon and the same script and the same size. This feature of *LinkWay* makes it easy to make a variety of objects of the same type.

As you have no doubt learned by this point, objects do not have to be named. However, to refer to them in scripts, you must give them a name. That name can be up to eight characters in length, including spaces. To use the name in a script, it must begin with a letter and include only letters, digits, and the underscore (_) character. Do not use spaces in object names that will be used in scripts.

You also need to be careful not to use the same object name more than one time on a page. If you use the same name for two different objects, *LinkWay* will not be able to tell to which object you are referring. In addition, do *not* use the name of any script commands or functions. Again, *LinkWay* may not be able to distinguish between the names you created and the names that are part of *LinkWay*. Duplicating names and using the names of *LinkWay* scripting words may cause strange bugs to appear that are difficult to correct.

Sometimes it is useful to know how objects are stored in *LinkWay*. When you are working with field objects, they are represented in the memory of the computer as a fixed-length string of characters. That length is the product of the number of rows and the number of characters in each row that is shown in the Field information window. This information allows you to tell how much text is contained in a field. Note that a field can hold from 1 to 20 lines of text and each line can be from 1 to 99 characters long.

If you are putting text into Text Pop-up buttons, Link buttons, Script buttons, or Reference buttons (see Chapter 8), the contents of these buttons is also represented as a string of characters. Text Pop-up and Script buttons can be up to 4,000 characters long, Link buttons can be 13 characters, and Reference buttons can be 16 characters. For more details, see your *LinkWay* documentation in the section on using script statements.

Object Order and Mouse Clicks

Objects on the page are "stacked" on top of one another as they are added to the page. A new object is put on top of an old one. However, the top-most object is not the "first" object on the page. The order of the objects on a page is the order in which they were created. Thus, the first object on a page is the first object that was created. You can check the order of the objects on the page by using the Tab key. First, be sure no object is selected. The first press of the Tab key takes you to the first object on the page. The second press takes you to the second object, and so forth. To change the order of the objects on a page, you can cut an object, delete it, and then paste it on the page again. When an object is pasted on the page, it becomes the last object on the page.

Recall that the object target area is the area of the object that responds to mouse clicks. There are rules that govern how mouse clicks are handled on a *LinkWay* page.

- If the mouse pointer is not in the target area of any object on the current page or the base page, a mouse click is ignored.
- If the mouse pointer is within the target area of an object, a mouse click activates that object.
- If the mouse pointer is within the target area of an object on the page and an object on the base page, a mouse click activates the object on the page.
- If the mouse pointer is within the target area of more than one object, the object activated by the mouse click depends on the object priority.
  - Fields take priority over pictures.
  - Buttons, media, and list objects take priority over fields and pictures.
  - If the objects with overlapping target areas are of the same type, a mouse click activates the object created first.

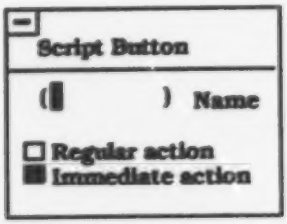
Scripts

Commands and functions are all used in the construction of *LinkWay* statements that are put together into small programs. In *LinkWay*, programming is called *scripting*. Each *script* is associated with a particular *LinkWay* button. The scripts you have seen so far are simply lists of *LinkWay* statements. *LinkWay* scripts can be divided into smaller parts called subroutines. You will learn more about these subroutines later in this book (Chapter 11). Often, script subroutines are grouped together and put on a separate page in the folder. Thus, scripts throughout the folder can refer to these separate little programs.

You have learned that script statements must end with a semicolon (;). While some commands work without the semicolon, leaving them out often leads to problems later. It is a good idea to *always* put a semicolon at the end of any statement.

The *LinkWay* special words that are the names for commands and functions are referred to in your documentation as *script statements* or *script functions*. To avoid confusion, the word *statement* is used in this book to describe a line of code beginning with a command and ending with a semicolon.

There are two types of scripts. Thus far you have seen only *regular action* scripts. These are scripts that are activated when you press and release the mouse button. There is a second kind of script, called an *immediate action* script. You have seen these two names in button dialog boxes:



Immediate action scripts are activated as soon as the mouse button is pressed. They do not wait until the mouse button is released. Immediate action scripts are used for actions like dragging an object while holding down the mouse button, flipping through the pages of a folder, or performing some action while holding down the mouse button.

Debugging and Testing

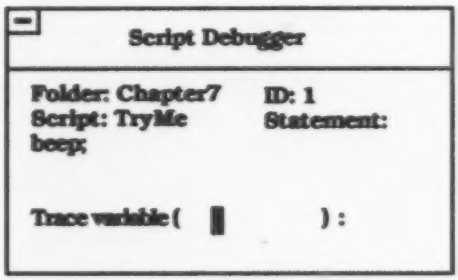
It can be very difficult to find the bugs in scripts. You have seen that you can use **msg** and **prompt** to have information displayed in the message windows or on the Status line. You have been reminded to be sure to spell the words in your scripts correctly and to put a semicolon at the end of every statement. You have seen how to use the Tracker tool to help you find screen coordinates.

Sometimes, however, all of these techniques fail. *LinkWay* has a tool to help with your debugging. Select **Debug** from the **Option** menu. A small window appears that allows you to turn on (enable) or turn off (disable) the debug feature. You can access this window only when there is no script running.

Put this script on a button of your choice:

```
beep;
wait 1;
beep;
beep;
wait 2;
beep;
beep;
beep;
```

Select **Debug** from the **Option** menu and click on Enable Debug. Click on the button. Now press the Scroll Lock key. You see a window something like this.



This window tells you the name of the folder (Chapter7), the Id of the button (1), the name of the button (TryMe), and the statement that was running when you pressed the Scroll Lock key. In addition, when you learn about variables in Chapter 23, you can enter a variable name and see its value as the script runs.

When you close the Script Debugger window, the script continues to run. As long as the script is running, each time you press the Scroll Lock key, the Script Debugger window appears. If the Scroll Lock key remains down, each time you run a new script the window appears. **Debug** can be quite a useful aid to finding problems in your scripts.

Project Ideas

Spend some time reviewing the commands and functions you have learned thus far. Does the discussion about *LinkWay* grammar help you put the special words used by *LinkWay* into patterns that you can better understand?

Write some simple scripts and practice using the **Debug** option. Pay careful attention to all of the information given in the Debug window. How can this feature be helpful when you are having trouble with a script?

Chapter 8. Document and Reference Buttons

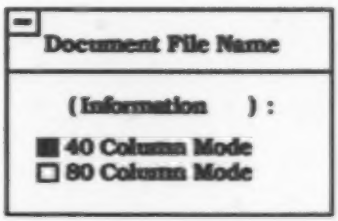
In this chapter, you will learn about the two kinds of buttons that allow you to access blocks of text with a mouse click. In addition, you will learn about using LWEdit to prepare text for these buttons.

Document Buttons

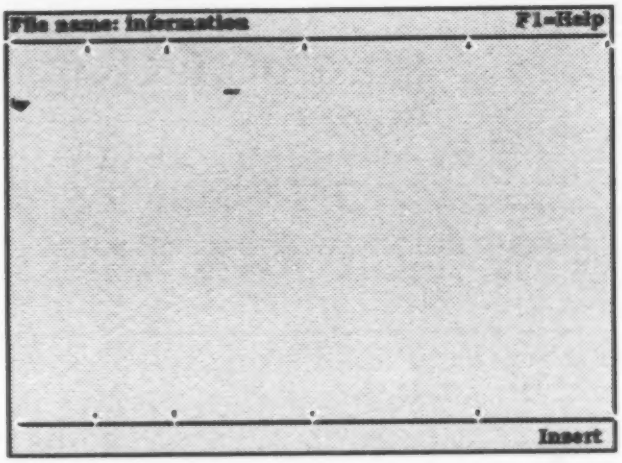
To begin your work for this chapter, start a new folder. You are going to put buttons in this folder that are connected to blocks of text.

Go to the **Object** menu and select **New**. Create a button of type Document. Document buttons look just like other *LinkWay* buttons, but they access a screen that looks quite different from the usual *LinkWay* screen.

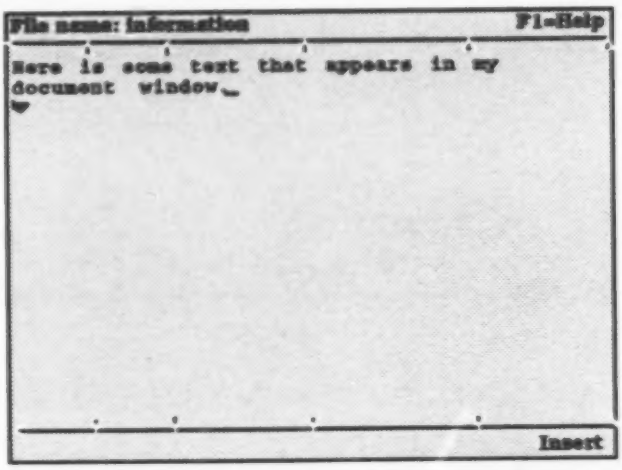
Name your new Document button Info. In the Document File Name box, select 40 Column Mode, and enter the name of the document that will be connected to the button. Call the document Information.



The Info button appears. Click on the button. You see a screen that looks like this:



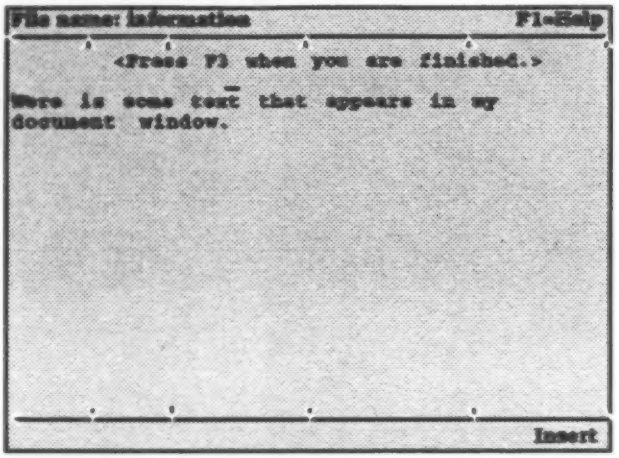
Enter some text.



When you are finished, press F3. You are automatically returned to the page in your folder containing the Info button. If you now click on the button, the document window automatically appears with the text you entered.

If you don't enter the name of the document when you create (or edit) the Document button, LinkWay *will ask you what document to edit each time you click on the button.*

Document buttons are useful for displaying large amounts of text on the screen. When you use document buttons, you need to tell your users what to do when they have finished reading the text on the screen so that they know how to return to your folder. They should not have to access the Help feature. Thus, you might modify the above page as follows:



Previously in this book you have used tools such as *LinkWay Paint* and Tracker buttons. Document buttons automatically access the tool called LWEdit. As you just saw, you can compose or edit documents in LWEdit. The keys used for editing are much the same as the editing keys discussed previously in this book (Chapter 3). A summary of the keys for working in LWEdit is given at the end of this chapter.

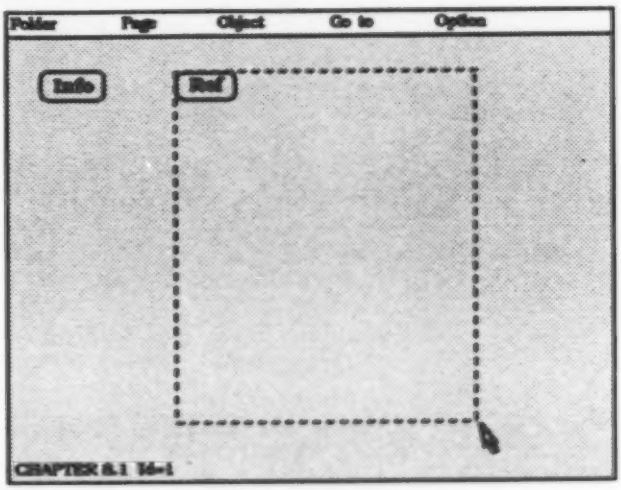
The files saved by LWEdit are text (ASCII) files. Text files from other word processors can also be used with Document buttons. See your *LinkWay* documentation for details.

Note that if the Access level of the folder is higher than Read, you can edit the text of a Document button. However, if the access level is Read, you can see the text but not change it.

Reference Buttons

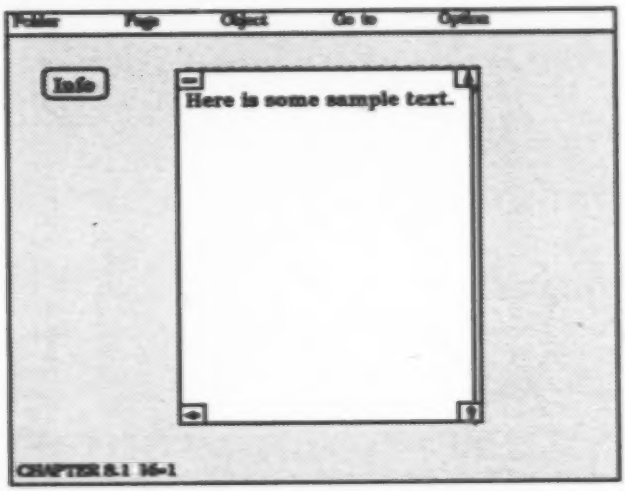
Reference buttons are similar to Document buttons in that they display text. However, Reference buttons differ in the way they display text. Document buttons provide a full 40- or 80-column screen. Reference buttons display text in a window on the page.

Begin by creating a Reference button. Call it Ref. Make sure the bounding box for the button is fairly large.



Call the reference document Reference. If you then double click on the Ref button, you'll get the message "Missing reference file."

Next, go to the **Object** menu, select **Tools**, and open LWEdit. Type the name Reference when asked. Be careful that your spelling is accurate. Enter some text in the Reference file. Press F3 to leave LWEdit. Now double click on the Ref button. You see



If you have a lot of text in a reference button, there are several ways you can move around in the text. If you wish to move through the text a line at a time, you can click the arrows in the scroll bar or use the arrow keys. If you wish to move a block at a time, you can click within the scroll bar above or below the scroll marker. You can also move through the text a block at a time by using the PgDn and PgUp keys.

Reference buttons are useful for displaying text in a window on the *LinkWay* page, but they also have another powerful capability. It is easy to produce what is known as *hypertext* using Reference buttons. Hypertext is text that is read in a nonsequential manner. Using *triggers* allows you can make particular words or groups of words behave much like Script buttons. Unlike Script buttons, these buttons move with the text even if it is scrolled. This kind of button is sometimes called a "sticky button."

To explore triggers, return to LWEdit and open the file that you called Reference. Select a word about which you want to include more information. Suppose you want to make the word "text" into a trigger. You must then type the correct characters before the word "text." Thus, the text with trigger added might look like this:

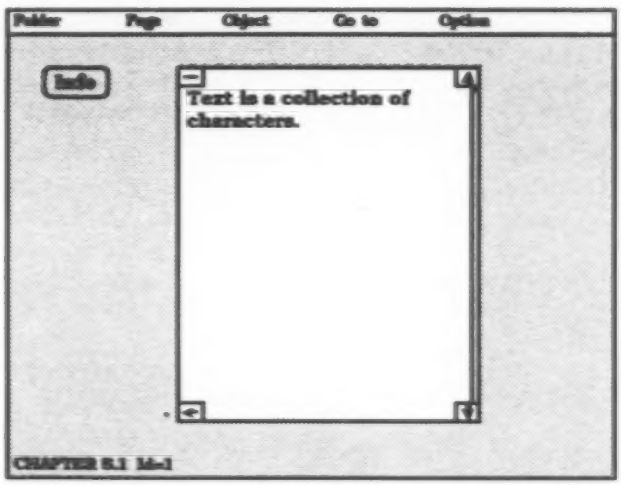
```
Here is some sample %F,15,MoreInfo%text .
```

The text shown in bold in the line above is called a *file trigger*.

Next use LWEdit to create a MoreInfo file. It might include the text

```
Text is a collection of characters.
```

Now if you click on the Ref button, you see the text of the Reference file displayed as usual. Most likely the word "text" appears in a different color. (If it doesn't, try a different number in place of the 15). Now click on the word "text." A second window appears:



You can click on the Close Dot to return to the page, or you can click on the arrow in the lower left corner of the window to return to the previous window of text.

File triggers can only be used on Reference buttons and must be written in a particular format. The syntax of a file trigger is

```
%F,color,filename%triggerword
```

Be careful to include all of the necessary information:

- % signs are used to enclose the information before the trigger-word.
- F specifies a file trigger.
- "color" is the color number of the trigger-word.

There are several other kinds of file triggers. Each is described briefly below. For more details, examine the section on Reference buttons in your *LinkWay* documentation.

Label triggers allow you to jump to a specific place in a file. The syntax of a label trigger is:

```
%#label%
```

where

- # specifies a label trigger.
- "label" is a string of eight letters and/or numbers.

Button triggers leave a reference file and select a button to be run. The syntax of a button trigger is:

```
%,B,color,buttonname%triggerword
```

where

- B specifies a button trigger.
- "color" is the color number of the trigger word.
- "buttonname" is the name of the button.

Exit triggers leave a reference file. The syntax is:

```
%E,color,returnstring%triggerword
```

where

- E specifies an exit trigger.
- "color" is the color number of the trigger word.
- "returnstring" is up to 30 characters that will be stored in the function **result** (see Chapter 17).

Return triggers return to the point in the calling reference file from which a file trigger was last run. The syntax is:

```
%R,color%triggerword
```

where

- R specifies a return trigger.
- "color" is the color number of the trigger word.

A color trigger changes the text color in reference text. The syntax is:

```
%C,color%
```

where

- C specifies a color trigger.
- "color" is the color number of the reference text.

Reference buttons are useful for providing access to a large amount of material from a single page. They can be used to allow readers to check on definitions of terms or find out more information about a particular topic. The use of the wide variety of triggers makes Reference buttons a powerful tool for developing *LinkWay* folders that are flexible and user-friendly.

Debugging and Testing

Document and Reference buttons provide a variety of options for giving the user of your folder access to large amounts of text information. However, creating these buttons means that you must be sure that all of the files you want to use with your folder are in the same directory. This can be a problem when you are using a program other than LWEdit to prepare your text.

Text in Document buttons is not limited in size. Users can scroll through the text using the scroll bar. Reference buttons are limited to 1,600 characters. If you have more than 1,600 characters in the file for a Reference button, it will be truncated the first time it is displayed.

As you are working with Reference buttons, it is easy to lose track of the names of files. Keep a pad of paper beside your computer as you work. Carefully write down the names of any files as you create them. It is also helpful if you give the files meaningful names. Don't be tempted to call your files A, B, and C to save typing!

Triggers used with Reference buttons are a powerful tool. However, you must take care when entering the text of a trigger. Minor typing and spacing errors will cause your triggers to function incorrectly. Check each trigger carefully as you add it to your Reference buttons.

Editing Keys for LWEdit

Key	Action
Arrow keys	Moves the cursor one character at a time
Home	Moves the cursor to the beginning of the line
End	Moves the cursor to the end of the line
PgUp	Scrolls text backward one screen
PgDn	Scrolls text forward one screen
Backspace	Deletes one character to the left
Ins	Toggles between insert and replace modes
Del	Deletes the character at the cursor position
F1	Help
F2	Saves a file
F3	Exits LWEdit and saves the text
Shift+F3	Exits LWEdit without saving
F4	Asks for the name of a text file and then copies it into the current text file
Shift + F4	Displays the screen in reference format with the correct lines lengths
F5	Searches for a text string
Shift+F5	Inserts the template string %F,color,filename#label% into the text
F6	Reflows a paragraph to the width of the screen
Shift+F6	Reflows a paragraph to the width indicated by the cursor location (minimum = 20 char.)
F7	Copies current line onto "pick" folder
Shift + F7	Inserts the top line from the pick folder
F8	Prints to a specified printer
F9	Deletes the line in which the cursor is located
Shift+F9	Joins the next line to the current one
F10	Inserts a blank line
Shift+F10	Splits a line

For more details on these key combinations, see your LinkWay *documentation on using LWEdit.*

Project Ideas

Spend some time exploring the concept of hypertext. Make a number of Reference buttons. Make the references several layers "deep" by using file triggers. Do you get confused or lost if you have too many layers of hypertext? How could you solve this problem?

Explore the use of the different forms of triggers. Can you make each kind work correctly? Do you see ways that they could be used in your work with *LinkWay?*

Create your own reference page. First create a number of text files containing notes and reminders to yourself. Perhaps they would include a list of commands and functions, keys to use for editing, and the like. Then create a reference page that uses Document and Reference buttons. Add this page to your previously created folder that contains your favorite buttons and fields.

Chapter 9. Design Issues

In this chapter, you will focus on the design of *LinkWay* folders. In particular, you will learn about folder structure, some ideas for making your folders look more professional, and some conventions you should take into consideration when designing folders.

The first eight chapters of this book have focused on the mechanics of writing *LinkWay* instructions and using *LinkWay* tools. You have learned how to write scripts for a variety of types of buttons. You have learned how to use *LinkWay Paint*. You have learned how to edit text in fields and scripts and how to use LWEdit to edit Document and Reference button documents. You have also taken a look at the structure of the *LinkWay* scripting language. Much of your work has been experimental, so you may not have created any particular product.

However, as you learn more about scripting in *LinkWay*, you will want to begin to apply your ideas to building folders that are of value to you and to others. *LinkWay* is more than a programming environment. It is a powerful tool for creating a graphical interface to work with information contained in folders. *LinkWay* is designed to help you create folders quickly and easily. However, it is also even easier to produce very badly designed folders. The user of such folders can easily become confused or lost. Paying careful attention to the design of your work is as important as the mechanics of writing scripts and creating pages.

Folder Structure

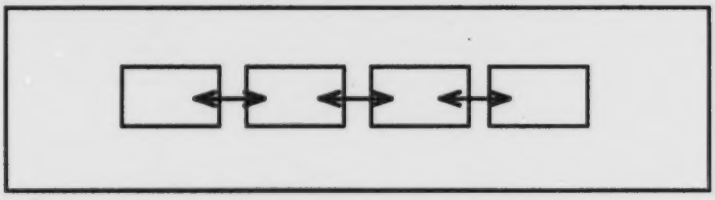
The first step in creating a *LinkWay* folder should be planning its structure. There are many ways you can put a folder together. One effective approach to folder design is to "outline" your folder on index cards. Using this method, you lay the cards out on a table to help you decide upon the links among your pages. You might also consider using a bulletin board to arrange the cards that represent pages in your folder. Post-it notes could be used to represent pop-up buttons, reference, and document buttons. You then use pieces of string or yarn to indicate links among pages. This approach lets you take a look at the overall structure of your folder and think through what the important relationships are among pages in your folder.

When designing a folder it is extremely important that you think carefully about the nature of the material you want to put into a folder. Is it "linear" in nature? That is, do you want your readers to see one page at a time in a set order, much as if they were reading a book? Or does the

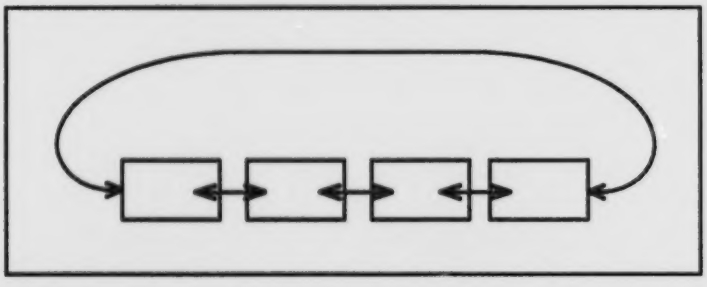
material "branch" naturally? Is it sensible for the user of your folder to want to explore one area, such as the habitats of animals, for a while and then be able to move with ease to the foods eaten by those same animals? Is your folder exploratory? That is, do you want users to be able to explore in a variety of ways and jump in no particular order from place to place among the pages in your folder? Answering questions such as these requires a lot of planning, examination of the material that is to become part of your folder, and thought about the nature of the final product. Not asking such questions, on the other hand, leads to products that are surprisingly difficult for others to use.

There are some general formats folders can take. These are discussed in the following sections. However, it is important that you *not* take these as rules but rather as general ideas about ways to structure folders. You may create folders that are exactly like some of the following models, or you may create folders that are totally different from any of these models. Or you might choose to use some combination of these ideas. The structure of your folder depends heavily on the nature of the material that makes up your folder.

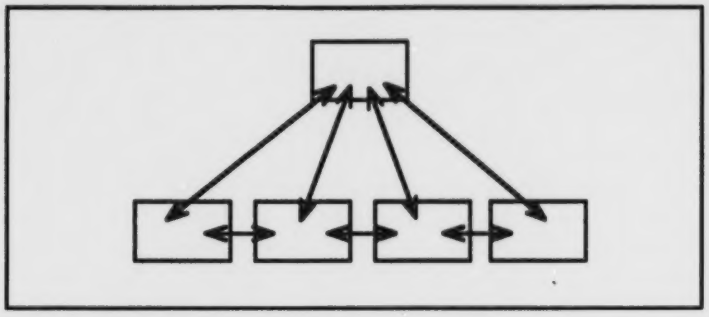
Folders may be strictly linear. You might use this type of folder if it is important that the user see each page in sequence. This structure might be used when you have a sequence of instructions and you want to be sure the user of your folder sees each instruction in order:



It is not uncommon for a linear folder to add a link between the first page and the last:

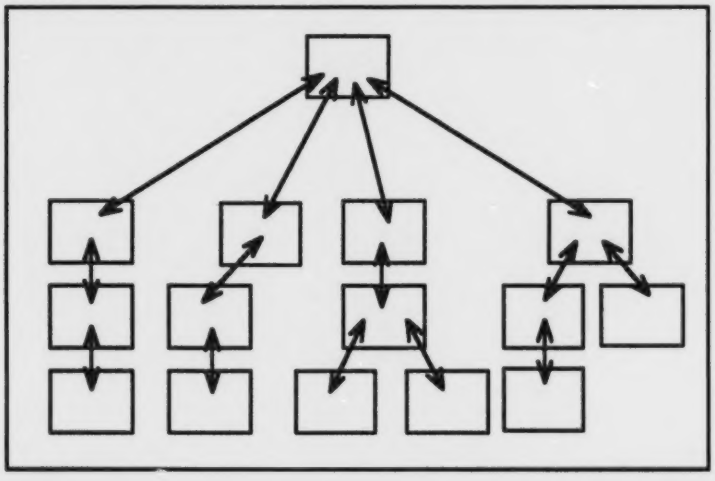


You might also want to make your folder "jump linear," allowing the user to jump from the "top" page to any page in the linear sequence or to move sequentially through the "bottom" row of pages:



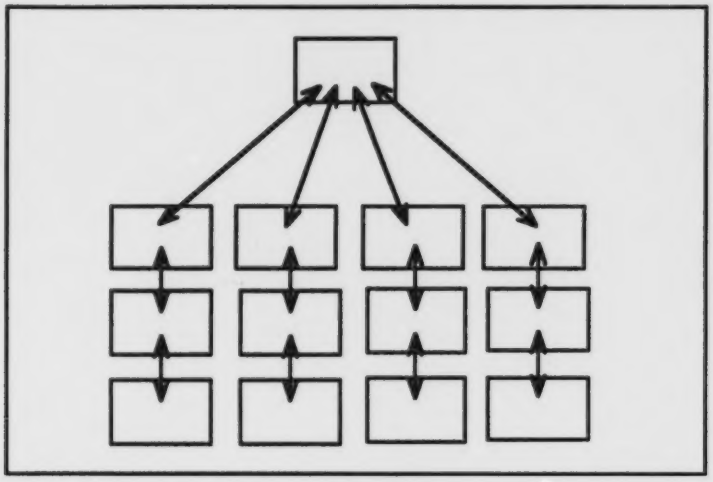
The jump linear model might be used to present a menu on the first page—the top box in the preceding diagram. The user can then go to any page in the linear sequence. Once in the linear sequence, the user can either go back to the menu or move back and forth in the linear sequence.

Computer scientists use the term *tree* to describe the structure shown in the following diagram:

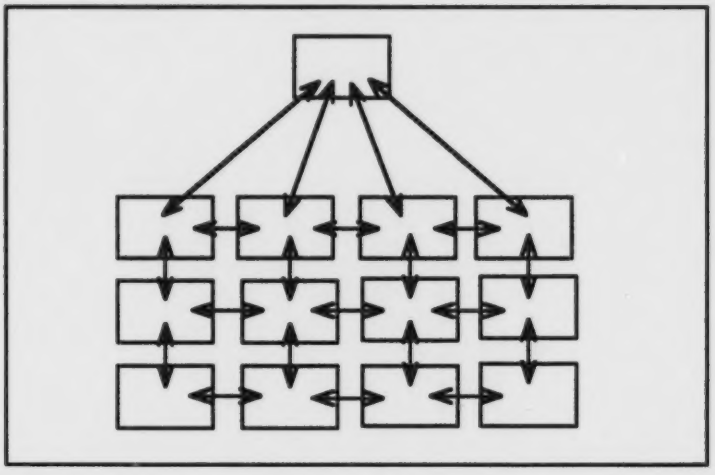


A tree structure allows users to move up and down branches of the tree to explore any particular idea in more depth. Tree structures are fairly natural ways to display information. For example, if your folder is about animals, you might have two subbranches: water animals and land animals. Then, in the category of water animals, you might have other divisions, such as vertebrates and invertebrates.

There is a temptation when working with tree structures to make as many links as possible among pages. Such an approach can be quite confusing to the user. The following tree structure is fairly easy to follow. Users know that once they are in a given branch of the tree, they will remain in that branch until they return to the top of the tree.



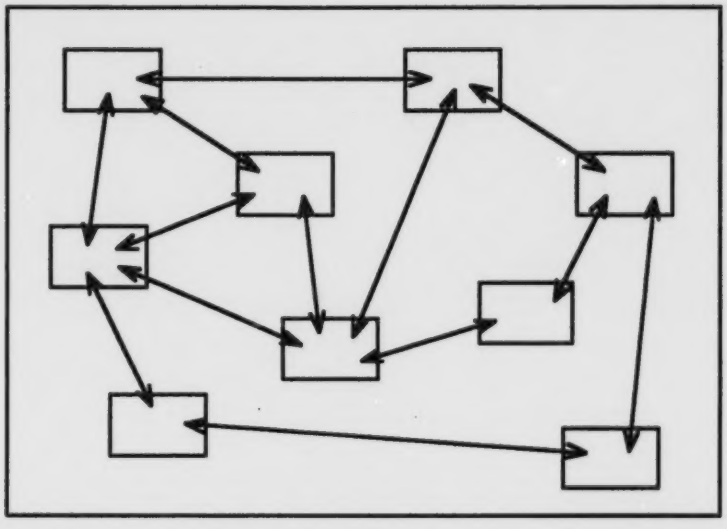
However, if you make many links that allow movement across the branches of the tree, users can easily become lost. The following diagram shows a structure in which it is easy to become confused or lost:



In this case, if users go down the "water animals" branch of such a tree, and then move to the right or left in the tree, they can easily find themselves in a "land animals" branch without being aware that the nature of the information being presented has changed. Making as many links as possible does *not* necessarily make your folder better. However, you need to be sure the links to the branches of a tree are bidirectional; that is, be sure the pages are linked so that a user can go down and then back up a given branch of the tree in some manner.

Finally, a network structure is used when you want the user to explore your folder in a variety of ways. Such folders present information that has no particular order to it. For example, you might represent airline routes

in such a folder. Or you might have a folder that describes various events during the Civil War. Or you might create an exploratory folder for children so that when they click on a picture of an object, they move to a related picture.



Although the model that we use to think about *LinkWay* folders is a file folder full of pages, not all folders look as if they have a lot of pages. "Single-frame" folders appear to have only one page. The user has no sense of moving through a number of pages at all. One type of single-frame folder is the "display folder." This type of folder is used to show a graphic or text field that changes depending on the user's choice. For example, you might create an alphabet folder for children. Along the top of the page there would be 26 link buttons, one for each letter of the alphabet. There would be 26 pages referenced by the buttons. Each page might contain a word beginning with the appropriate letter and a picture associated with the word. When the child clicks on the button for a letter, a picture of an object beginning with that letter appears. In such a folder, the pictures might appear in a standard "frame" on the page so that there would be no sense of movement.

Another form of single-frame folder is a "filter folder." This type of folder may serve as a kind of menu for a single large folder or "front end" for another group of folders. Filter folders are particularly useful when a folder becomes quite complex and contains a lot of information. For example, a folder that contains information about all of the major countries of the world might filter the information by allowing the user to choose to look at, for example, African countries only.

A third type of single-frame folder is a "point-and-click" folder. Such folders let the user click on various items on the page to learn more about them. For example, a folder that presents a map of a city might give brief

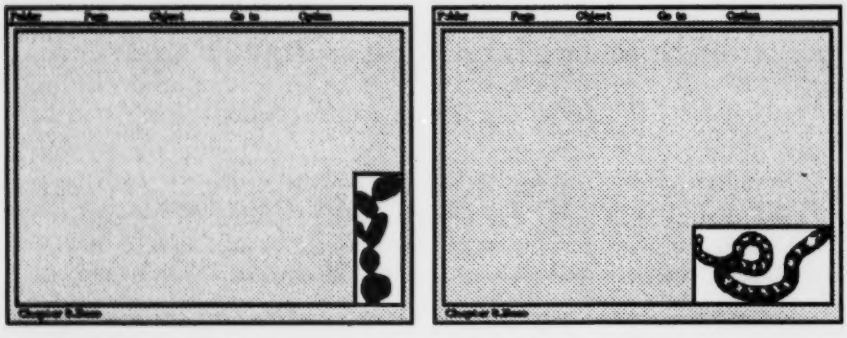
descriptions, addresses, and phone numbers of restaurants when the user clicks on the appropriate picture on the map. In such a single-frame environment, the user would have no sense of moving to another page; rather, the information would appear to "pop up" on the original map page.

Base Pages

Some beginning users of *LinkWay* don't understand the power of using base pages. They simply ignore the blank base page *LinkWay* puts into a new folder. They forget that they can put objects on the base page. They end up putting "next" and "previous" buttons on every page, forgetting that they can simply put those buttons on the base page. Or they cut and paste pictures or re-create frames or borders on every page. None of these approaches makes appropriate use of *LinkWay* base pages.

The base page you choose for your folder should reflect the content of your folder. It certainly is inappropriate to have a base page that looks like index pages if your folder is supposed to look like a slide show. You should give a good deal of thought to the base page as you design your folder.

The base page can serve as a visual cue for users of your folder. If you are developing material in *LinkWay* about several related subjects, for example, desert plants and desert animals, you should consider creating two different folders with distinct base pages.



A third "menu" folder can be used to link to the two other folders.

You should always use the power of base pages. If you have a button or field that needs to appear on every page, put it on the base page. If there are just a couple of pages that don't use that particular button or field, you can cover the base page objects with invisible buttons when needed. And putting objects on the base page makes every page look more consistent. For example, a button will appear to be in exactly the same position on every page if it actually is on the base page. This avoids the problem of objects "jumping around" as the user flips through your

pages. Think about redesigning your base page if you find yourself putting the same object on page after page.

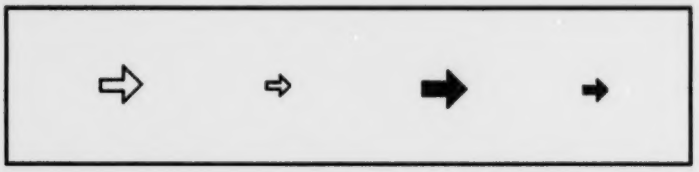
Finally, designing your base page is as important as designing the structure of your folder. Carefully thinking about what should be on the base pages in your folder in advance of developing your folder will save you a lot of time at the computer keyboard.

Conventions for Using Icons

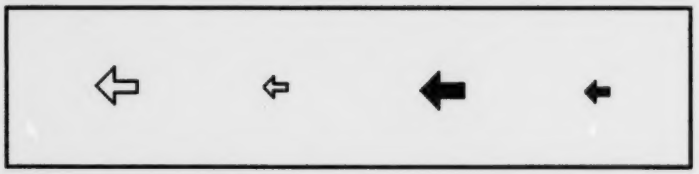
There are some conventions for using various icons on buttons. If you deviate too much from these conventions, users who have used other *LinkWay* folders may have a great deal of difficulty getting around in your folder. Recall that there are a number of icons available in the Button Icon window and that you can also design your own icons.

If you design your own icons, you must be very careful. A small icon that has meaning to you may not have meaning to other users of your folder. Be sure to have a variety of people look at your icon to be sure that they understand it. Similarly, if you use standard icons, they need to be used in fairly standard ways.

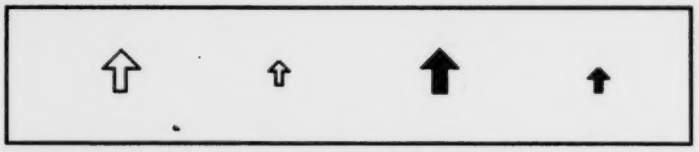
For example, the following icons standardly mean "go to the next page" in the folder:



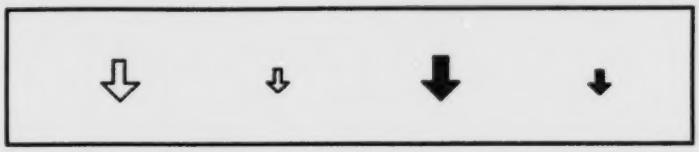
On the other hand, the next group of icons usually mean "go to the previous page":



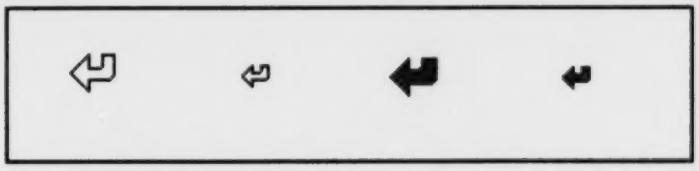
In addition, the following up arrows generally mean "go to the previous page":



Down arrows generally mean "go to the next page":



The following set of icons commonly means "return." Return can have a lot of different meanings in *LinkWay*. Return can mean "go back to the menu," "return to the last branch in the folder," and so forth.



An icon that looks like a small house often means "go to the Main folder." There are also icons for Help and Go available in the Button Icon window.

Some Other Considerations

It is often a good idea to include a button that links back to the Main folder. It might be on every page, or at least on pages that are easily reached at any time when using your folder. It is very easy for novice users to become confused when using *LinkWay*. Allowing them to return to the Main folder gives them a way to start over.

You should think carefully about the Access level you want built into your folder. An experienced *LinkWay* user will be able to modify the Access level by using the menus as long as you don't include a password. However, a novice user won't be able to make such changes. If it is important that novice users not be able to make any changes in your folder, set the user level quite low. If you don't want to allow advanced users to modify the folder, include a password when you set the Access level. Just be sure you carefully record your password for your own use later.

Another important consideration when designing folders is consistency. If buttons and fields are not placed carefully, the transitions between pages will be unpleasant. Buttons and fields may appear to jump around if they are not at exactly the same spot on each page. Using the Alt key while moving or placing objects will help you to line them up along character grid lines. Furthermore, putting objects used on most pages on the base page ensures that the objects will be at the same place on every page in the folder.

Even worse than having buttons appear to shift slightly as you move through the folder is having buttons that accomplish the same task

appearing at different places on every page. This forces users to search for buttons they need on each page.

If you use icons on buttons, make every effort to be sure that the meaning of the icon is clear. Even better, put the icon on the button and include a word in a field below the button to make the functions clear.

Consistency and clarity in placement and use of the various elements of your folder are quite important. Keep in mind that *you* are very familiar with the design of your folder. A new user is not. You must be sure that everything on your folder is easy to understand.

You should also think carefully about whether you want the Menu bar visible. If you expect your folder to be completely self-contained, you probably want the Menu bar hidden. Similarly, you may want to hide the Status line. The Status line can be distracting to a novice user, especially when it displays information that is generally not relevant to the content of the folder.

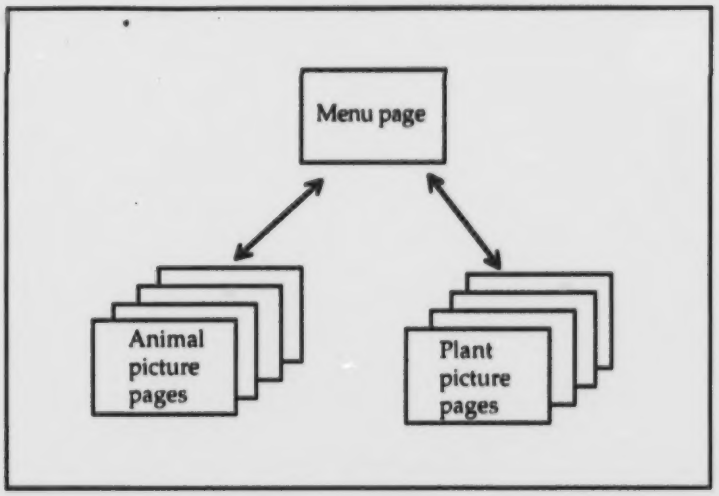
To hide the Status line, select **Status** from the **Option** menu. To turn the Status line on again, simply select **Status** again from the **Option** menu. To hide the menu bar, select **Menu Bar** from the **Option** menu. The Menu bar disappears. To display the Menu bar again, move the cursor to the top of the screen and click in the area where the **Option** menu was. The menu will drop down



and you can select **Menu Bar** to again display the Menu bar.

Debugging and Testing

It is easy to lose track of the structure of your folder as you are working, even if you planned it carefully in advance. To help visualize your folder, it is a good idea to make a map that represents your folder. A folder map looks like the diagrams at the beginning of this chapter, but with individual pages identified. For example, you might have the following folder map:



As you develop increasingly complex folders, it can be easy to get lost among your pages as you work. You can retrace your path through your folder by using **Retrace** from the **Go to** menu. Each time you select **Retrace**, you move back one page in the pages you have accessed. *LinkWay* keeps track of the last nine pages you have accessed. If you have not gone through nine pages, *LinkWay* only steps back as far as you have previously gone during your current session with the folder.

Project Ideas

Many of the ideas briefly summarized in this chapter are discussed more extensively in a variety of sources. Spend some time reading more about the design of hypermedia products, either in books or magazines. Think more about the issues involved in creating good folders.

Examine some existing folders. Are they easy to understand? Why or why not? See what kinds of overall folder design you can find, and think about the ones that would be most useful to you.

Have someone else criticize the design of some folders you have created. Do they see your folder differently than you do? How can you make your work more user-friendly?

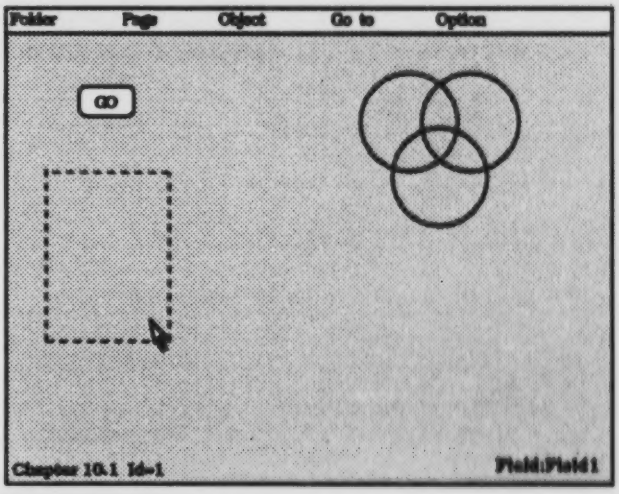
Chapter 10. More on Working With Objects

You have learned to work with buttons, fields, and pictures from scripts. In this chapter you will learn more about the properties of objects and will learn some other commands that allow you to manipulate objects from scripts.

Each of the objects you put on a page has properties, such as its name and its position, associated with it. You have used the name of objects in a **set** command, for example, to put text into a field. You have used the coordinates of the bounding box of an object to move it with **move** or **wmove**. *LinkWay* contains a number of functions that report such information as the name and position of an object. This information can be used to work with objects from scripts.

Object Properties

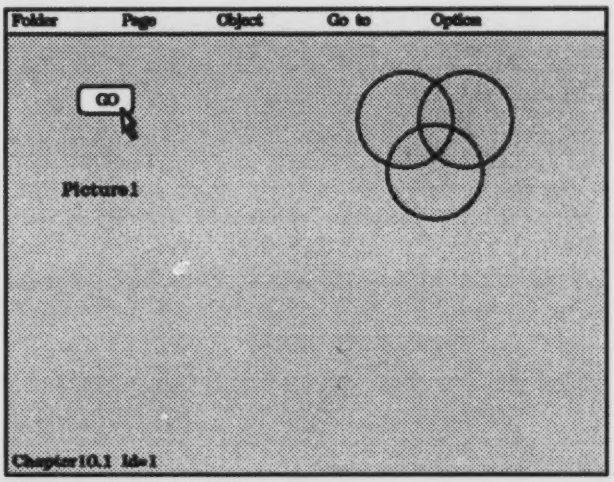
Begin by creating a new folder. On the first page of the folder, make a Script button called Button1 with the GO icon from the Button Icon window on it. Leave the script empty for now. Next, make a field called Field1 and place it below the button on the page. Finally, create a picture called Picture1 and put it on the right side of the page. It is not important what you use for a picture; however, it should not be large.



The function **oname** returns the name of the object. Put this script on the GO button:

```
mouse;
set Field1 = oname;
```

Now, double click on the GO button. Then click once on the picture. The name of the picture appears in Field1.



Recall that **mouse** waits until you click on an object. Also recall that when **set** is followed by a field name, it puts the text after the equal sign into the field. The function **oname** returns the name of the selected object. The syntax of **oname** is:

```
oname
```

Try clicking on the other objects on the page. Remember, because you are at the Format Access level, you must click twice on the button and then click only once on the object you want to select. If the objects are named carefully, this technique could be used to create a page for small children to use. Pictures of household objects, animals, or toys could be put on the page. Clicking on an object would then show the correct spelling of the word for that object.

When a button has an icon on it, the function **oicon** returns the number of that icon. Change the script on the GO button to

```
mouse;
set Field1 = oicon;
```

If you then click on the button twice to activate it and then once more to select it, the number 9 appears in the field. The number of the GO icon in the Button Icon window is 9. The syntax of **oicon** is:

```
oicon
```

Note that if a button has a *name* instead of an icon, 99 appears in the field. If the button has neither a name nor an icon, the number 0 appears in the field.

There is also a function, **otype**, that returns the type of the object. Its syntax is:

```
otype
```

If you modify the previous script to

```
mouse;
set Field1 = otype;
```

and then click on an object, the type of the object will be displayed in the field. If an object is not selected, **otype** returns a blank. The following list gives the values that are returned by **otype**.

F	Field
BD	Document button
BF	Find button
BG	Go Button
BL	Link Button
BP	Pop-up button
BS	Script button (regular action)
Bs	Script button (immediate action)
BT	Text Pop-up Button
P	Picture
LinkWay Live! only:	
BM	Media object
L	List

When you create a field or a pop-up button, it has a width measured in number of characters. The function **owid** returns the width of the object. If you have the script

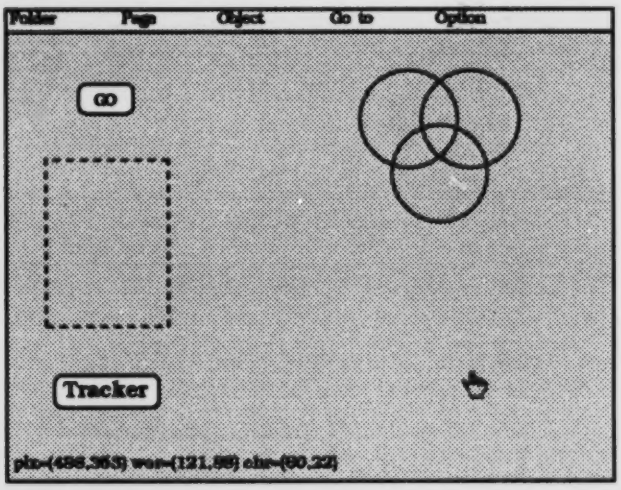
```
object "Field1";
set Field1 = owid;
```

the width of the field appears in the field.

Coordinate Systems

You have encountered three coordinate systems in your work with *LinkWay*. Sometimes positions are given in column-and-row character positions, sometimes in pixels, and sometimes in world coordinates. Recall that the Tracker tool (Chapter 5) can be used to display the mouse location in all three coordinate systems. Add a Tracker button to the page you are working with. Move the mouse about the page to see how the values change.

The display mode affects the range of values of these three coordinate systems. However, the largest column-value in the *character* coordinate system will be 39, 44, or 79, while the maximum row value will be either 24 or 29. The *world* coordinates are similarly consistent. The horizontal coordinate varies from 0 to 159 while the vertical coordinate ranges from 0 to 99 or from 0 to 119. If your mouse pointer is in the lower right part of the screen, you might see



The specific ranges for character coordinates and world coordinates for each mode can be found in your LinkWay *documentation under* ***move*** *and* ***wmove****.*

The pixel system represents the number of dots on your screen and is much more variable from mode to mode than the world or character systems.

Using Positions of Objects

There are four functions that can be used to determine the character coordinates of the currently selected object. These functions are **oleft, oright, otop,** and **obot**. Modify the GO button so that its script reads as follows:

```
object "Picture1";
set Field1 = oleft;
```

When you click on the button, something like

```
45
```

appears in the field, indicating that the left side of the bounding box for Picture1 is at the character coordinate of 45. This information might be clearer if you modify the script as follows:

```
object "Picture1";
set Field1 = "Left = ":oleft;
```

If you click on the button now, you see

```
Left = 45
```

Notice the colon (:) between the quoted text and the **oleft** function. Using the colon concatenates—puts together—two strings of text.

If you further modify the script, you can display all of the coordinates:

```
object "Picture1";
set Field1 = "Left = ":oleft:"Right = ":oright:"Top =
   ":otop:"Bottom = ":obot;
```

If you click on the button, the character coordinates appear. Unless your field is exactly the right size, all of the text may not appear as you would expect. Later in this chapter you will learn how to correct this problem.

If you are having difficulty with syntax errors when working with the long lines such as the one in the previous script, remember that after you make changes in the line, you need to use F9 to "glue" the pieces of the line back together.

There are two parts to solving the text placement problem that may have occurred in the previous script. First, you can used **flowset**. Recall that **flowset** causes the text to flow smoothly into the field:

```
object "Picture1";
flowset Field1 = "Left = ":oleft:"Right =
   ":oright:"Top = ":otop:"Bottom = ":obot;
```

Perhaps you would like for each coordinate to appear on a separate line. You can use the **chr** function to accomplish this task. The syntax of **chr** is:

```
chr(number)
```

Each key on your keyboard has what is called an ASCII (ask' ee) value. The ASCII value is a number that represents this character in the memory of the computer. The number following **chr** must be an ASCII value.

The ASCII value 13 represents a carriage return. That is, inserting an ASCII value of 13 will cause the text to go to the next line. The **chr** function can be used to accomplish this task:

```
object "Picture1";
flowset Field1 = "Left = ":oleft:chr(13):"Right =
   ":oright:chr(13):"Top = ":otop:chr(13):"Bottom =
   ":obot;
```

The syntax of these four functions for finding the coordinates of the bounding box is:

```
oleft
oright
otop
obot
```

Remember that these functions return the coordinates by using the character grid.

You can use these functions to cause an object to move when a button is clicked:

```
object "Picture1";
move oleft, otop + 10, oright, obot + 10;
```

The picture moves 10 rows *down* the screen. A series of such statements can be used to cause an object to move across the screen.

LinkWay Live! has a second set of functions that return the *world* coordinates of the bounding box of an object. These four functions are **wtop, wbot, wleft,** and **wright.** Their syntax is:

```
wtop
wbot
wleft
wright
```

When you use the functions that return the character coordinates (**otop, obot, oleft, oright**), the object you are working with must be selected. However, the current object is assumed when using the world-coordinate functions.

The Mouse Position

Sometimes it is useful to know the current location of the mouse. You can use the functions **mousex** and **mousey** to get this information. The syntax for these two functions is:

```
mousex
mousey
```

Change the script on your button so that you see

```
flowset Field1 = mousex:",":mousey;
```

Then click on the button. You see the location of the mouse in *pixels* when you clicked on the button. Note that the comma in quotation marks is concatenated—put together with—the coordinates so that an ordered pair appears in the field.

If you modify the previous script so that it reads

```
mouse;
flowset Field1 = mousex:",":mousey;
```

the command **mouse** makes you click on the screen in order to see the coordinates. The coordinates of the mouse click appear in the field.

Did you note that the **mouse** functions return the coordinates in pixels? Since most of the other functions give values on world or character systems, you need the functions **pxwx** and **pywy** to convert pixel coordinates to world coordinates. Their syntax is:

```
pxwx
pywy
```

Thus, to get the world coordinates of the mouse, you can modify the previous script as follows:

```
mouse;
flowset Field1 = pxwx(mousex):",":pywy(mousey);
```

Debugging and Testing

As you develop increasingly long scripts, you will have more places where bugs can occur. There are a number of strategies for dealing with bugs. It was previously noted that you need to use F9 to put together lines that have become separated as you edit scripts. *LinkWay* will give you a syntax error if the lines are not rejoined, even if your typing is exactly correct. When you get an error, be sure to check that all of the appropriate lines are joined.

Next, carefully check your printed references, such as the Appendix of this book, for the syntax of the statement. Should you have quotation marks? Should there be spaces? Did you forget the semicolon at the end of lines? Did you insert the wrong punctuation? Did you omit some necessary punctuation?

If all of the previous suggestions fail to correct your error, try testing your script one line at a time. For example, you might move the first line to the end of your script. Does the second line have a syntax error? You will find out when you run your script. You can cycle through each line of your script in this manner without having to erase your work. You can use

F7 to copy the line onto the stack of stored lines, or you can put it onto the stack with F10. Then use Shift + F7 to put the line at the end of the script.

Recall that you can also use the Debug tool. Select **Debug** from the **Option** menu. You can then use the Scroll Lock key to turn the Debug function on and off.

New *LinkWay* Words in This Chapter

Functions	
`chr(number)`	`otype`
`mousex`	`owid`
`mousey`	`pxwx`
`obot`	`pywy`
`oicon`	`*wbot`
`oleft`	`*wleft`
`oname`	`*wright`
`oright`	`*wtop`
`otop`	

*Available only in *LinkWay Live!*

Project Ideas

Spend some time with Tracker buttons becoming more familiar with the three coordinate systems. Make a chart of the maximum horizontal and vertical values of each coordinate system for each mode you use.

Use the world coordinate functions to move an object.

Try using the mouse to select a location on the screen and then have an object move to that location.

Create a page with a variety of pictures on it. Create a matching set of buttons. Have each picture move in some interesting way when you click on the button.

Chapter 11. Writing and Using Procedures

In this chapter you will learn how to break your *LinkWay* programs into small parts by using procedures. You will then learn how to use these procedures to make your *LinkWay* code easier to read and debug.

Since very early in this book, you have worked with a variety of scripts. Most of these scripts have been fairly short. As you develop your skills with *LinkWay* scripting, you will find that your scripts can get long and quite difficult to understand.

Dividing your scripts into procedures helps you to write code that is easier to work with because:

- You can make your code easier to read.
- You can avoid writing the same commands again and again.
- You can more easily test parts of your code.

The following examples explore each of these ideas.

Making *LinkWay* Code Easier to Read

One of the most important reasons for learning to write procedures is to make your *LinkWay* code easier to read and understand. One way to accomplish this task is to *modularize* your scripts. Modularizing your scripts means breaking them into small, meaningful parts.

Suppose you have a button that moves a picture when you click on it. The picture moves to a new location, you hear two beep sounds, there is a pause for a second or so, the picture then moves back to its original location, and again there are two beeps. The script to accomplish this task might look like this:

```
object "DemoPict";
move oleft, otop + 10, oright, obot + 10;
beep;
beep;
wait 1;
move oleft, otop - 10, oright, obot - 10;
beep;
beep;
```

While this script isn't particularly complex, we'll use it as an example to show how you might modularize longer and more complicated scripts in the future.

First, create a button called MoveFd. On that button put this script:

```
proc;
move oleft, otop + 10, oright, obot + 10;
return;
```

The command **proc** tells *LinkWay* that the code that follows is a procedure. The **return** command tells *LinkWay* that it has reached the end of the procedure and should return to the main program.

Note that because procedures have no names within the script—their names are given on Script buttons—there can be only one procedure per button.

The syntax of **proc** is:

```
proc [(value, value,…value)]
```

You will learn more about using values after **proc** in Chapter 25. The syntax of **return** is:

```
return value
```

You will also learn more about using a value with **return** in Chapter 25.

Create two more buttons, MoveBk and Sound. On the MoveFd button, put this procedure:

```
proc;
move oleft, otop - 10, oright, obot - 10;
return;
```

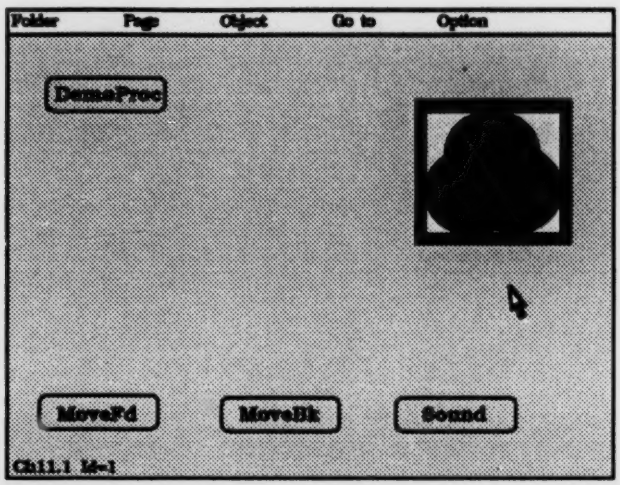
On the **Sound** button put

```
proc;
beep;
beep;
return;
```

Put the following code on the main button:

```
object "MyPict";
call MoveFd;
call Sound;
call MoveBk;
call Sound;
```

Finally, add a picture called MyPict to your page. Your screen now looks something like this:



Now when you click on the DemoProc button, you see the picture move and hear two beeps. There is a pause, the picture then returns to its original location, and there is another pair of beep sounds. The command **call** tells *LinkWay* to use the code on the procedure buttons. The syntax of **call** is:

```
call scriptButtonName [(value, value…value)]
```

You will learn more about using values with **call** in Chapter 25.

Note that you cannot click on the procedure buttons to run them. These buttons simply "hold" procedures that are called from a "main" program. If you click on the procedure buttons, you get a script syntax error.

In general, when you are working with scripts using several procedures, the buttons that contain those procedures are not visible to the user. These procedure buttons might be invisible buttons on the current page. Or you might set aside a page for procedure buttons elsewhere in your folder. The general rule of thumb for *LinkWay* scripting is to put your procedures on a page or two at the beginning of the folder because procedure buttons are accessible from anywhere in the folder.

Another way to make your procedures more readable is to add comments to your code. Comments in *LinkWay* scripts are called *remarks*, and the symbols `/*` and `*/` are used to enclose the comments. Adding comments helps those reading your scripts to understand them better. In fact, you will find that it is easier to read your own scripts later if you have included remarks. For example, the previous script that moved a picture might look like this if remarks were included:

```
/*This script moves a picture on the screen.*/
object "MyPict";
/*Moves the picture down 10 character positions.*/
move oleft, otop + 10, oright, obot + 10;
beep;
beep;
wait 1;
/*Moves the picture up 10 character positions.*/
move oleft, otop - 10, oright, obot - 10;
beep;
beep;
```

Even when you have used procedures to modularize your scripts, remarks can help understand your work:

```
/*Moves a picture up and down the screen*/
object "MyPict";
call MoveFd; /*moves down 10 characters*/
call Sound;
call MoveBk;/*moves up 10 characters*/
call Sound;
```

It is a good idea to include comments in all of your work. A comment at the beginning of a script should tell the purpose of the entire script. Comments throughout the script should clarify parts of the code that are not immediately understandable.

Using Procedures More Than Once

Another reason for modularizing code is that particular procedures can be used more than once. In the previous example, you saw that the Sound button was used twice.

Recall that the ***drowcol*** *command in available only in* LinkWay Live!

Suppose you want to be able to place messages on the top left, top right, or bottom center of the page. You might create three procedure buttons. On MsgLeft you could put

```
proc;
drowcol 5,5;
return;
```

On MsgRt you could put

```
proc;
drowcol 60,5;
return;
```

and on MsgCen you could put

```
proc;
drowcol 40,20;
return;
```

Recall that **drowcol** places the box for a message. Thus, on any other button on the page, you might have

```
call MsgRt;
msg "You win!"
```

or

```
call MsgRt;
msg "Great job!";
```

Any one of the procedure buttons can be used any number of times. Using this technique involves writing less code, thus avoiding typing and editing errors you can easily make while entering complex *LinkWay* statements.

Testing Your Scripts

A third important reason for using procedures has to do with testing. You have developed a number of techniques for debugging your scripts. Working with procedures is another technique you can use. Suppose you want a procedure to move a button across the screen. Begin by making a script button call MoveObj and entering this script:

```
object "BtnArrow";
move oleft + 5, otop, oright + 5, obot;
```

When you are certain that this code works, add the following lines:

```
object "BtnArrow";
move oleft + 5, otop, oright + 5, obot;
wait 1;
move oleft + 5, otop, oright + 5, obot;
wait 1;
move oleft + 5, otop, oright + 5, obot;
```

Be sure to use the editing keys to copy and paste the identical lines. Now test the script.

When you are sure this script works properly, modify the code so that the button becomes a procedure button:

```
proc;
move oleft + 5, otop, oright + 5, obot;
wait 1;
move oleft + 5, otop, oright + 5, obot;
wait 1;
move oleft + 5, otop, oright + 5, obot;
return;
```

You can now use this script from any button on the page or even elsewhere in the folder. To test your procedure, make a new Demo button and enter this script:

```
object "BtnArrow";
call MoveObj;
```

Test the new button to be sure it works. Make other buttons that use this script. Test them. Note that you do not have to rewrite a lot of code to

move any number of objects across the page with the button procedure MoveObj.

Debugging and Testing

Using **Debug** from the **Option** menu when you begin to use procedures can be very helpful. Because **Debug** displays the name of the object and the line being run, you can tell which button contains any lines that have problems.

Make frequent use of remarks as you are creating scripts. They will help you understand your own thinking as you debug and modify them. When you are finished with a script, take time to be sure the remarks are clear and easy to understand. That way, if you or anyone else needs to read the script at a later time, it will be much easier to modify or correct.

When you are designing your folders, you should think carefully about how you might modularize your code. Where will procedure buttons that are used throughout your folder be put? Which procedure buttons should remain on the page and which should go on a separate page?

New *LinkWay* Words in This Chapter

Commands
`call` *`scriptbuttonname`*
`proc`
`return` *`value`*

Project Ideas

Create some simple handlers of your own. A good place to start is with some code to move objects. Divide the move commands into small pieces and test each one. Then use the button procedures in a main procedure.

Look at some of your previous scripts. Are there places where you could modularize your work? Are there places where you could make your work clearer by using meaningfully named button procedures? Would the addition of remarks make your work clearer? Take some time to "clean up" a couple of your earlier projects.

Create a "library" of button procedures that you think you might use often. Be sure to give them meaningful names and add remarks to the procedures so that when you use them in different scripts you can tell exactly what they do.

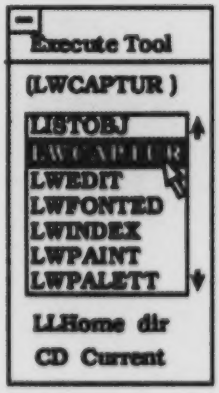
Chapter 12. Using LWCapture

You have learned to use a variety of tools with *LinkWay*. For example, you have used *LinkWay Paint* to create graphics for your folders and you have used LWEdit to edit text used in Document and Reference buttons. In this chapter you will learn to use yet another tool to import graphics from other sources into your *LinkWay* folders.

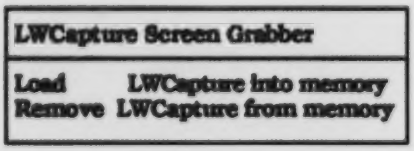
You have been using picture objects in your *LinkWay* folders since early in this book. In some cases you have used pictures that came with *LinkWay*. In others you have created your own pictures by using *LinkWay Paint*. In this chapter you will learn how to take graphics images, including digitized images, commercial clip art, and images produced in other paint programs, from a variety of sources.

LWCapture

To begin, select **Tools** from the **Object** menu and choose LWCAPTUR.



You then see the following window:



Click on Load to load the program into memory.

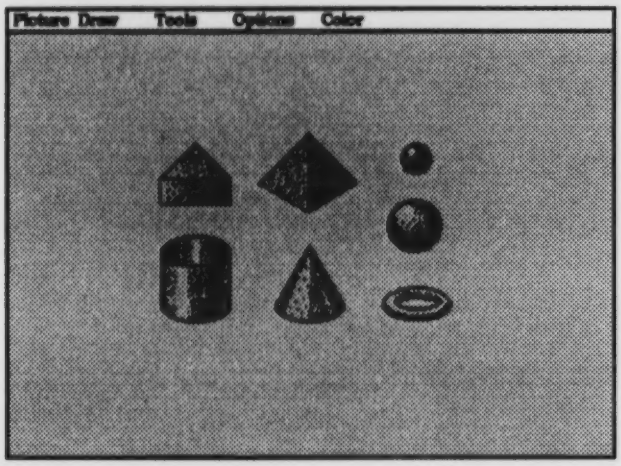
LWCapture is a memory-resident utility. It remains in memory, available for use until you remove it. You can also use LWCapture outside of

LinkWay. *See the chapter in your documentation that discusses capturing and converting pictures.*

There will be a pause while the program is loaded into memory.

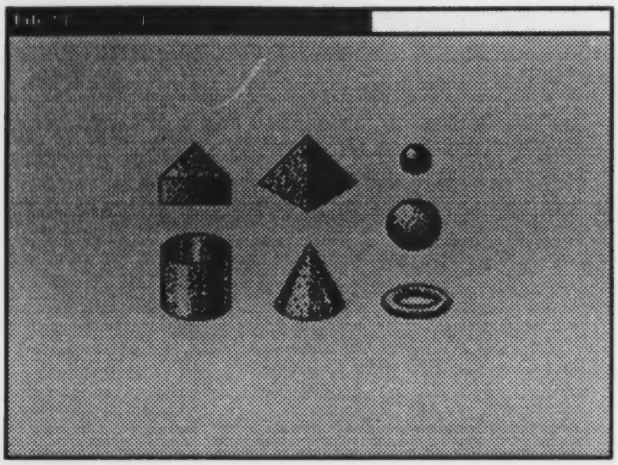
Next, you need to locate a picture you want to import to a *LinkWay* folder. Locate some graphics that came with your version of *LinkWay*. (Note that we are using *LinkWay Paint* to demonstrate the use of LWCapture because there is no is to guarantee that every reader of this book has access to any other graphics program.)

To begin, open the *LinkWay Paint* program and open a picture of your choice.



Note that if you are using a program different from LinkWay Paint, *you follow the same procedure. You must first display the picture on the screen by using appropriate software.*

Next, press Shift + PrSc. (Your key may be labeled Print Screen instead of PrSc.) A prompt appears at the top of the screen.



You can either type the name of your picture or use the default names. (The default names are PIC000, PIC001, PIC002, etc.) Your picture is now stored under a new name. You can then use the picture in *LinkWay*.

> *Note that the display mode used to capture the picture must be the same as the display mode for your* LinkWay *folder. LWCapture automatically captures a picture in the format currently being used by* LinkWay Paint.

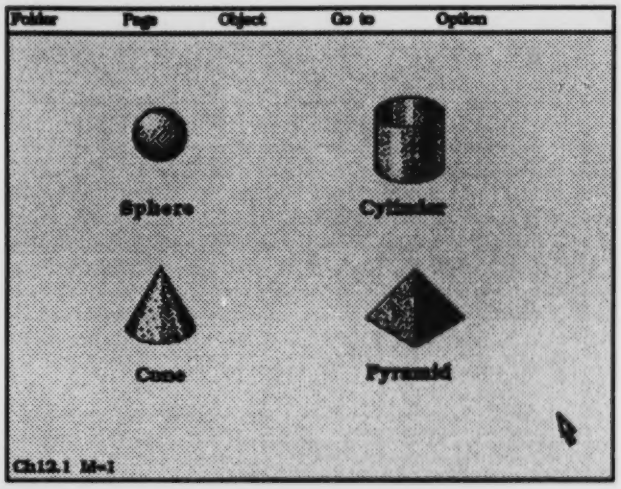
Next, exit *LinkWay Paint*, return to *LinkWay*, and open the picture you just saved. Of course, you can use *LinkWay Paint* pictures directly in *LinkWay*. However, the process just described works for any graphics you can display on the screen.

> *If you want to use some other art, see your* LinkWay *documentation under LWCapture. Also, read the section at the end of this chapter called "Some Technical Matters."*

Remember to choose **Tools** from the **Object** menu, select LWCAPTUR, and remove LWCapture from memory when you finish using it.

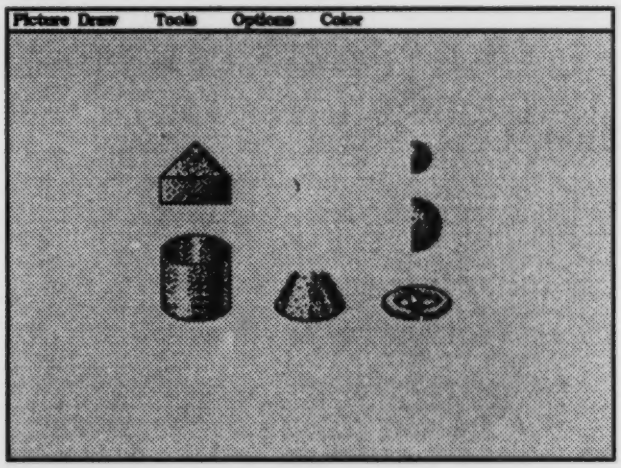
Using Captured Images

As you have learned, you can display only part of a picture in *LinkWay* by defining a bounding box smaller than the picture and using the Adjust window to place the picture so that the appropriate part of the picture shows on the screen. By using this technique, different elements from the same picture can be used several times. For example, you might use the preceding picture several times to produce a page that looks like this:



When you save a picture that contains multiple images, like the six shapes shown in the original picture, you save disk space. However, this strategy slows down the speed at which the picture is displayed. Thus, you may have to choose between speed of display and disk space used.

Another way to save disk space is to black out parts of a larger image and then save the picture. In *LinkWay Paint*, you can use **Bar** from the **Draw** menu. Select the background color and draw a bar over the parts of the picture that you don't want to use.



You can use several bars until only the part you want remains. But be careful! Be sure you have carefully saved your original image and that you save your modified images with a new names!

When you are preparing graphics to use in your *LinkWay* folders, you need to think carefully about how you will be using the pictures later. Name the pictures meaningfully so that you can tell what files you have.

You might even want to keep a log of the images you have. If you are working with a lot of graphics, you might consider printing them and putting them into a notebook. Be sure to write the name of each file on the printed image.

Some Technical Matters

LinkWay folders can access pictures with a .PCx extension. That is, you can use pictures with the extensions .PCC, .PCM, .PCE, .PCV, .PCH, .PCJ, and .PCZ. If a program saves pictures with a .PCX extension, you must rename the picture with the appropriate extension for the display mode in which it was produced. Note that *LinkWay* can only display pictures in the mode for which it was created.

There are three conversion utilities provided with *LinkWay*. These utilities can be accessed by clicking the Tools button on the opening page of the Main folder. (You can get to this page by selecting **Main** from the **Folder** menu.)

> *If you are using* LinkWay *rather than* LinkWay Live!, *you do not have the following utilities. The only conversion tool available is a utility for converting* IBM Storyboard *files, called SBConvrt. It is accessed from the LinkWay Tools on the opening page of the Main folder. .*

The utility PCX2LW converts many types of .PCX files to *LinkWay* format. Click on the Help button to get some assistance with using the program. Click on the Doc button to see the full documentation. Click on **Main** on the **Folder** menu to return to the Main folder.

Similarly, the AVC2LW and the STORY2LW utilities can be reached from the Tools button on the opening page of the Main folder. The AVC2LW converts AVC files while the STORY2LW converts *IBM StoryBoard* files. Both these utilities have Help and Doc buttons to provide assistance.

Project Ideas

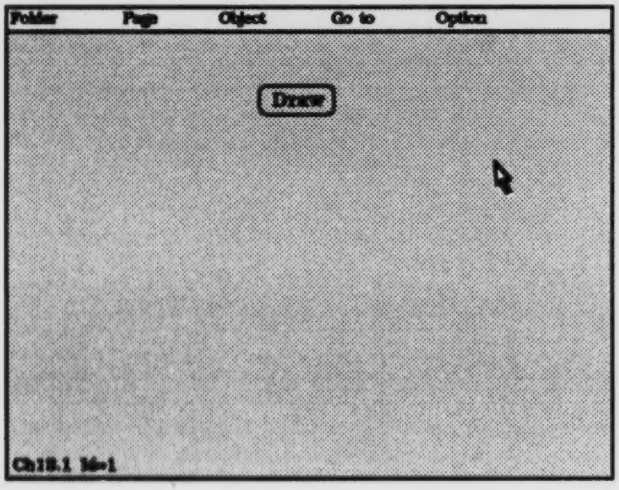
Spend some time becoming familiar with LWCapture. Practice saving some images and putting them into a *LinkWay* folder.

Explore the sources of pictures that you have available. Learn how to convert any clip art you may have to a format usable in your folders. Try creating or modifying a picture in your favorite painting program and then importing the image into *LinkWay*. You can explore using LWCapture to capture the images or using the conversion utilities to convert the pictures you save to *LinkWay* format.

Chapter 13. Working With Graphics From Scripts

You have learned how to create graphics by using *LinkWay Paint* and to use graphics created in other software in *LinkWay* folders. You have used the graphics options from the **Page** menu to create lines, boxes, and bars. In this chapter you will learn to use scripts to create simple graphics on the page.

Begin by creating a page containing a script button called Draw.

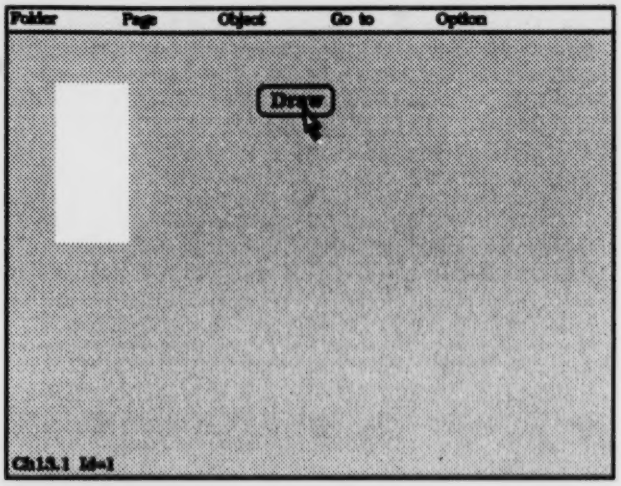


Using Bar and Box

Put the following script on the Draw button:

```
bar 1,50,50,100,200;
```

Click the button. You see something like this:



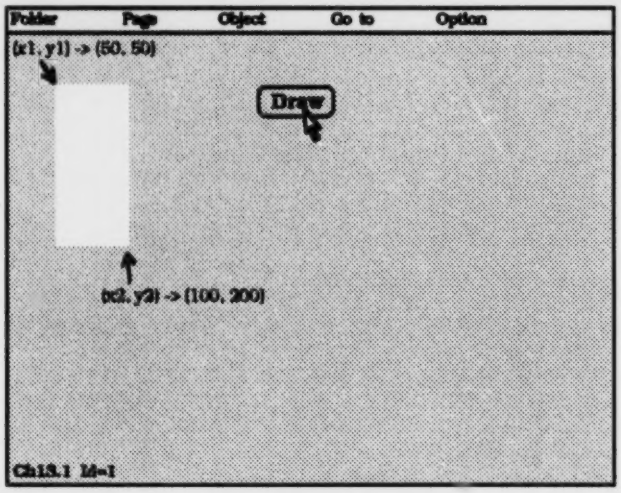
The syntax of the **bar** command is:

```
bar color, x1, y1, x2, y2
```

Color is any color number in the available range of colors for your display mode.

See your LinkWay *documentation for the list of color numbers available for your display mode.*

The coordinates (x1, y1) and (x2, y2) represent the upper left and lower right corners of the bar.



These coordinates are given in the *pixel* system.

Recall that the pixel coordinate system may be different for your system than for the example in this book. Change the numbers if necessary to make the graphics fit on your screen.

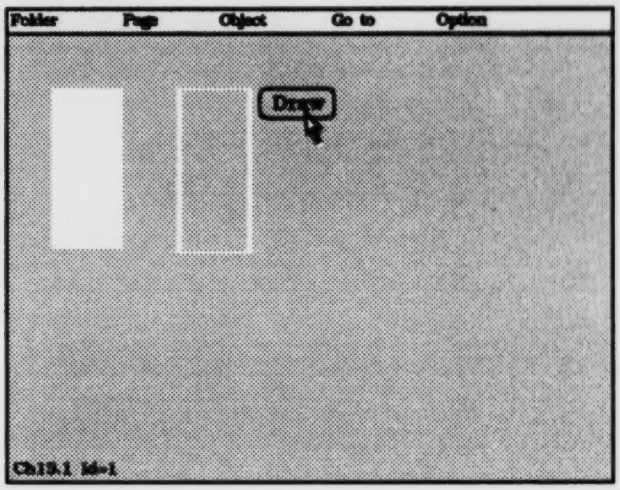
The **box** command works much like the **bar** command. Its syntax is:

```
box color, x1, y1, x2, y2
```

Modify the script on the Draw button as follows

```
box 1,150,50,200,200;
```

You then see something like this:



Using Line

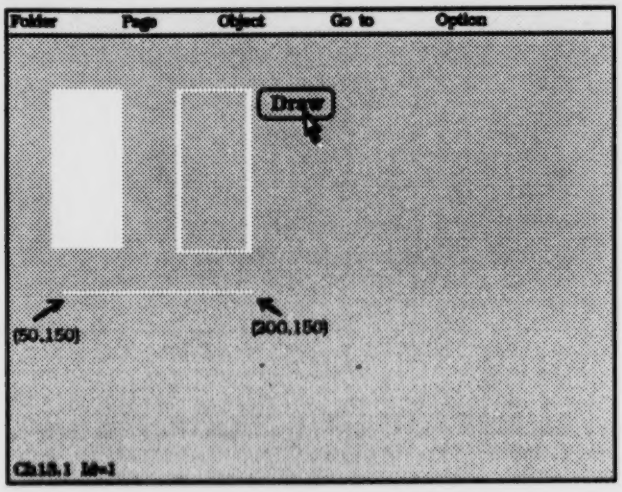
The **line** command is similar to the **bar** and **box** commands just described. Its syntax is:

```
line color, x1, y1, x2, y2
```

In this case, the first two coordinates describe one end of the line, while the second two describe the other. For example,

```
line 1,50,150,200,150;
```

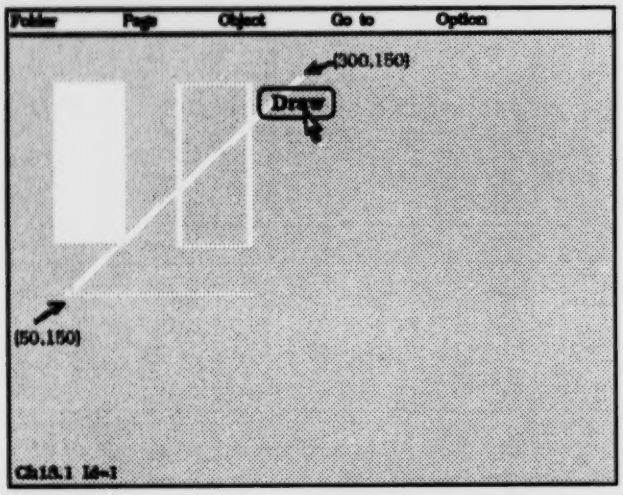
draws a horizontal line:



while

```
line 1,50,250,300,50;
```

draws a diagonal line:



Using Color Commands for the Background

There are a number of commands for working with color from scripts. The command for changing the background color is **bg**. Its syntax is:

```
bg colornumber
```

where "number" is a number from the following list:

0 black	5 purple/magenta
1 blue	7 white
2 light green	8 gray
4 red	14 rust brown

These color numbers apply no matter what mode you are working in, unless, of course, you are using Monochrome mode.

You can, for example, create a script that causes the background to flash by writing

```
bg 8;
wait 1;
bg 0;
wait 1;
bg 8;
```

The background changes from gray to black and back to gray.

If your are using *LinkWay Live!*, you are not limited by the colors listed above. Color number 14 can be changed by using the **brgb** command. The syntax of this command is:

```
brgb red, green, blue
```

where red, green, and blue are numbers between 0 and 63. These numbers represent the relative amounts of each of the three colors—red, green, and blue. The new color created by **brgb** is then assigned to color 14. Thus, to create a background color that is a mix of red and blue, but that includes no green, you can write

```
brgb 20,0,20;
```

Then, if you add

```
bg 14;
```

to the script, the background takes on your new color.

Using Color Commands for Objects

There are also color commands that are used to change the colors of other objects on your pages. The command **tcolors** is used to change the colors of Text Pop-up buttons, script windows, list objects, and Reference buttons. The syntax is:

```
tcolors foreground, background, window
```

Each of the three inputs to **tcolors** takes a number in the available color range. The third input, window, is the color of the outline of the box and of the scroll bar.

The command **dcolors** affects the colors of the text and background of boxes such as those produced by **msg**.

*The command **dcolors** is only available in* LinkWay Live!

The syntax of **dcolors** is:

```
dcolors foreground, background
```

where foreground and background are the appropriate color numbers. The default numbers are 8 for the foreground and 7 for the background. Thus, to change the color of the text, you might have the following:

```
dcolors 14,7;
msg "What is your name?"
```

The text appears in color number 14, while the background of the window is color number 7.

Similarly, **icolor** changes the color of buttons and button icons.

*The command **icolor** is only available in* LinkWay Live!

The syntax is:

```
icolor number
```

The default number is 15 (white) unless the background is white, in which case the default is 0 (black). Thus, to change the color of a button, you use

```
icolor 5;
```

The button is now color number 5. Note that button colors do not change until they are redisplayed. That is, you must leave the page and return to see the changes, or you can add "go seq" to the script to force the page to redisplay. (You will learn more about **seq** in Chapter 15.)

Debugging and Testing

Note that there is no "erase" script command corresponding to **Undraw** on the **Page** menu that you can use to erase boxes and bars you have drawn. To remove such features from your pages, you need to reuse the commands with a color choice that matches the background. That is, if you used

```
bar 1,50,50,100,200;
```

to draw a bar, and if the background is color number 0, you can use

```
bar 0,50,50,100,200;
```

to remove it. In Chapter 25, you will learn to use parameters to generalize statements such as this.

As you are working in *LinkWay*, are you remembering to save frequently? It is quite easy to become engrossed in your work and forget to save. If some problem occurs, like a loss of electrical power or accidentally erasing something you wanted to keep, you can easily lose many hours of work. You should save your work at least every 15 minutes or so. Furthermore, it's a good idea to save your work under different names. Suppose you are working with a folder called MyWork. Then you should save the folder the first time as MyWork1. The next time you save it, name it MyWork2. Continue this sequence every 15 minutes or so. If you don't like some recent changes you have made, you can always open a previous folder (earlier in your number sequence) and make different modifications. When you are finished working for the day, you will probably want to erase the oldest folders. However, it's a good idea to keep your several most recent copies in case you want to backtrack when you begin again.

New *LinkWay* Words in This Chapter

Commands
`bar color, x1, y1, x2, y2`
`bg colornumber`
`box color, x1, y1, x2, y2`
`*brgb red, green, blue`
`*dcolors foreground, background`
`*icolor number`
`line color, x1, y1, x2, y2`
`tcolors foreground, background, window`

*Only available in *LinkWay Live!*

Project Ideas

Spend some time exploring the use of **box**, **bar**, and **line** from scripts. Here are some ideas.

- Create a bar graph that appears one bar at a time.
- Create a folder with buttons such that when you click on them, a box that matches the name of the button appears around a picture.
- Create the axes for a graph using **line**.

- Script a button so that it creates a simple drawing, one part at a time.
- Use **bar** to highlight some text. Put some text in a field and then use **bar** to emphasize some part of it.

If you have access to color, spend some time experimenting with color for the various objects on your page. How can you use color effectively to enhance the attractiveness of your folder? Be careful not to overdo colors. What seems exciting the first time may quickly become overwhelming, annoying, or boring when seen frequently. After you have created a folder with colorful objects, have a friend critique your work for its design.

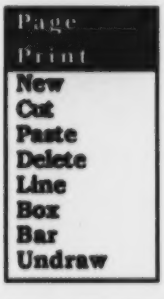
Chapter 14. Printing

This chapter gives a brief overview of printing using both the options available from menus and printing from scripts.

In general, documents produced using *LinkWay* are designed for use on the computer. It is virtually impossible to reproduce on paper the nonsequential nature of many *LinkWay* folders. However, there are times when you want to print part or all of your folder. While the strength of *LinkWay* is not in its printing capabilities, it does provide a variety of printing options.

Printing a Page

If you want to print a single page from your folder, you can use the **Print** option from the **Page** menu.



The current page will print in black on white on a graphics printer. Foreground colors will be printed as black. This includes everything on the page, such as lines, text, pictures, and icons. Note that this implies that text printed over any color on the background will be black-on-black and therefore unreadable.

If you are using a laser printer, the page will print in up to 16 shades of gray. In general, pages printed on a laser printer are much more readable than those printed on a dot matrix graphics printer.

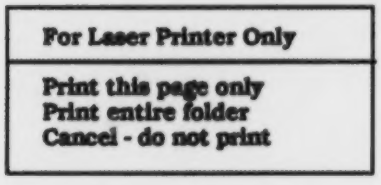
Printing Using Tools

LinkWay provides two tools for printing pages and folders. If you are using a dot matrix printer, select **Tools** from the **Object** menu and then select DOTPRINT. The following window appears:

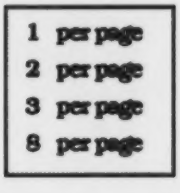


Use the mouse to select the option you want. Note that you can cancel the printing by selecting the last option.

If you are using a laser printer, select **Tools** from the **Object** menu and select LASPRINT. A window similar to the preceding one appears:



If you choose to print a page or to print the entire folder, you see



Select the number of page-images you want to appear on the printed page. The more images per page, the smaller the image.

There are details about the specifications for using these print tools in your LinkWay *documentation.*

Printing Pictures

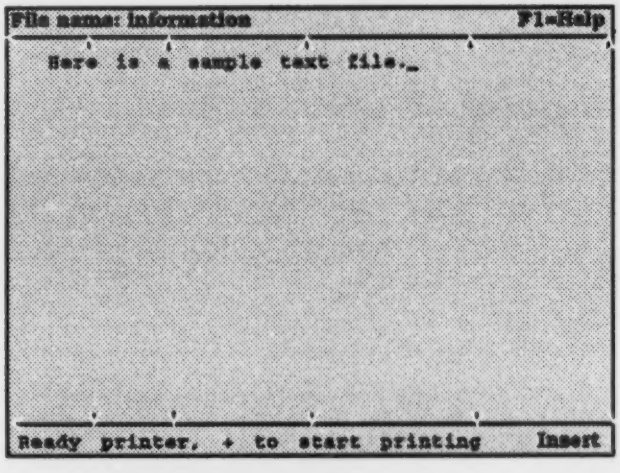
You can print individual pictures using *LinkWay Paint*. Select **Tools** from the **Object** menu and select LWPaint from the list. Next, use the **Picture** menu to open the picture you want to print. Finally, select **Print** from the **Picture** menu.



Your picture will be printed in black and white on a dot matrix graphics printer or in shades of gray on a laser printer.

Printing Text Files

Any time you are using LWEdit, you can print your text file. When you have opened your text file in LWEdit, press F8. You see a message at the bottom of the page:



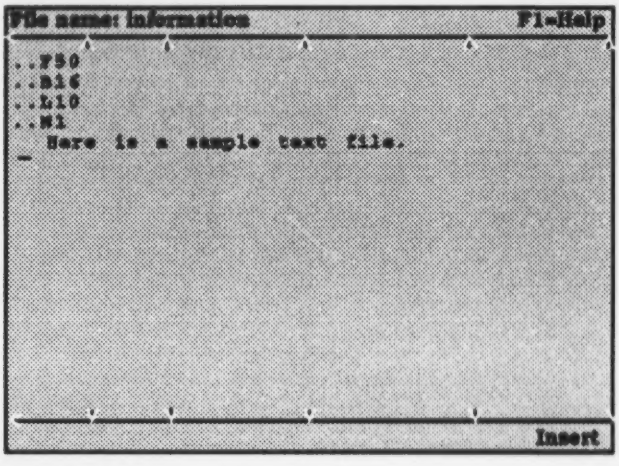
Make sure your printer is ready to print. Then press the + key.

You can modify the way text prints from LWEdit by using dot commands. You insert these dot commands into your file. Each dot command must be on a line by itself.

..Fn	Sets the page length to n lines. The default is 66.
..Bn	Sets the bottom page margin to n line. The default is 11 lines skipped at the bottom of each page
..Ln	Sets the left margin to n spaces. The default is 0 spaces.
..Nn	Begins page numbering with n. Setting n to 0 turns off page numbering.
..P	Skips to the top of a new page.

Note that each command begins with two periods. Note also that if a dot command is *not* included, the default value is used.

The following example prints pages that are 50 lines in length. There are 16 blank lines at the bottom of the page, and 10 blank spaces at the left side of the page. The pages are numbered beginning with 1.



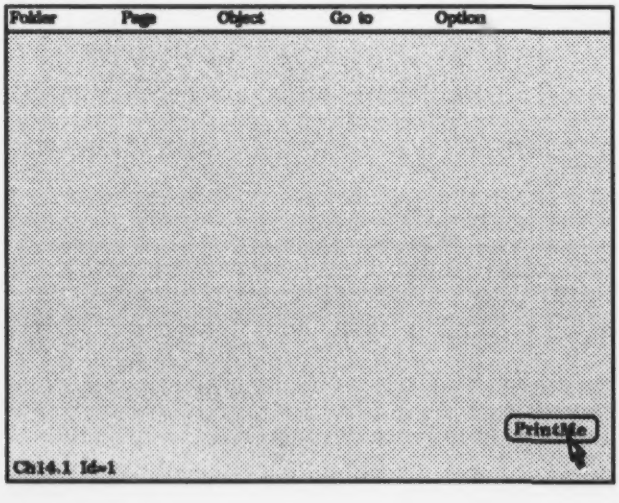
Note that these dot commands can be inserted throughout the text file to change any of these values. Note that you must insert the new page dot command at the point at which you want to begin a new page. Be sure that each embedded dot command starts on its own line.

Printing From Scripts

There are two commands, **print** and **cprint**, you can use to print information in your folders from scripts. The **cprint** command prints the current page, while the **print** command is used to display text messages on the printer.

You can add a script button called PrintMe to your folder with the following script:

```
cprint
```

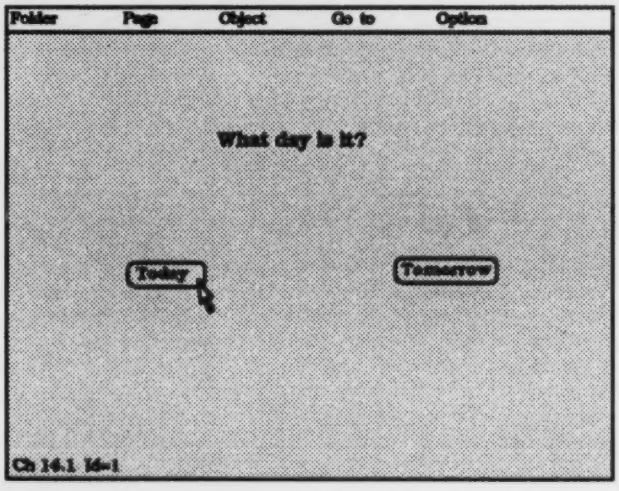


When the user of your folder clicks on the button, the current page prints, using the printer that has been set up to work with your copy of *LinkWay*.

The **print** command is used to send a string to the printer. Its syntax is:

```
print expression
```

where expression is a string containing information to be printed. Suppose you have a "quiz" on a page of a *LinkWay* folder:



The script on the Today button might read

```
print "Yes, it is today!"
```

while the script on the Tomorrow button might read

```
print "No, it's not tomorrow yet!"
```

This is a rather silly example, but you can see that you can use **print** to produce written reports of the progress that a user makes through your folder.

New *LinkWay* Words in This Chapter

Commands
cprint
print *expression*

Project Ideas

Spend some time experimenting with the various ways that you can print. Examine the documentation that came with your version of

LinkWay. Do you understand the differences among the various ways to print? Experiment with each of them until you are sure you know how to use them.

Chapter 15. Animation

In this chapter you will explore three different ways to create animation. You will use some techniques you already know and will learn a new technique for "flipping through" a number of pages.

You have already created some simple movement in your work with *LinkWay* scripting. For example, you have seen that you can use **move** and **wmove** to cause pictures and buttons to move across the page. There are a number of other techniques you can use to cause the effect of movement on the page.

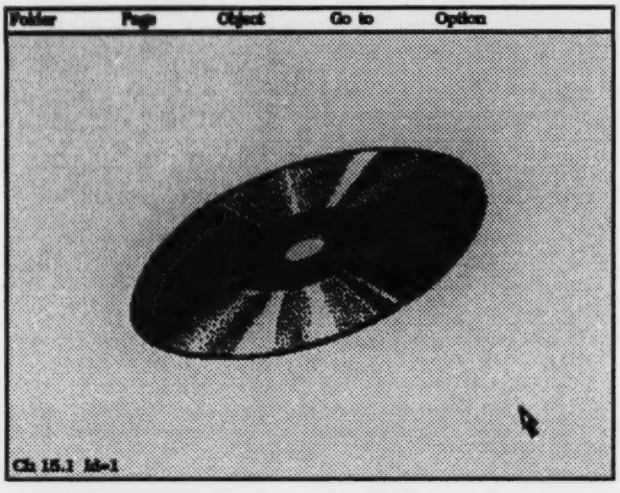
If you are animating small pictures, buttons are often a good choice to use to create the animation. For example, suppose you want to create the effect of having a telephone ring. First, in the Button Icon window locate the number of two telephone icons that are different sizes. (Of course, you can create your own telephone icons if you wish.)

Make a new Script button called Phone, with one of the phone icons on it. Then enter this script:

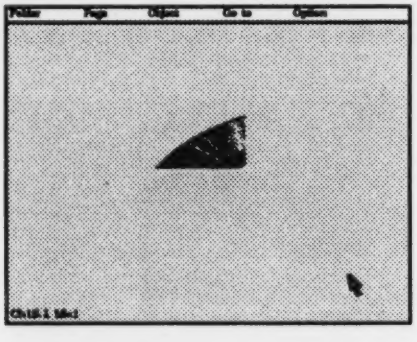
```
object "Phone";
beep;
change 61;
wait .5;
change 49;
wait .5;
beep;
change 61;
wait .5;
change 49;
wait .5;
beep;
```

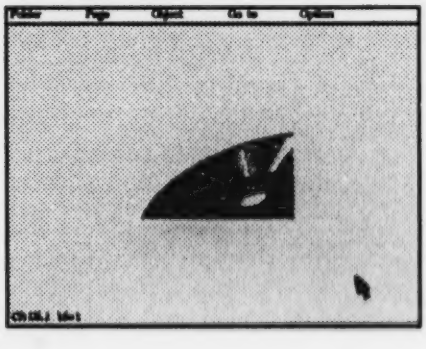
Now when you click on the button, the phone rapidly gets larger and smaller, giving the effect of movement. If you have access to sounds other than the beep sound, you might want to add an actual ring sound to your phone button.

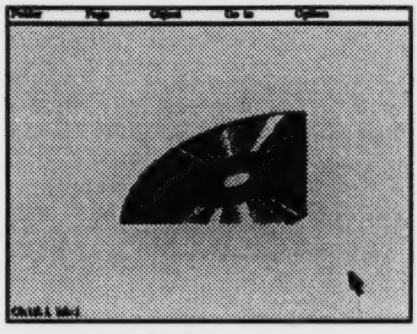
You can also create some interesting effects by using picture objects. Suppose you have a picture that fills a large part of the page.

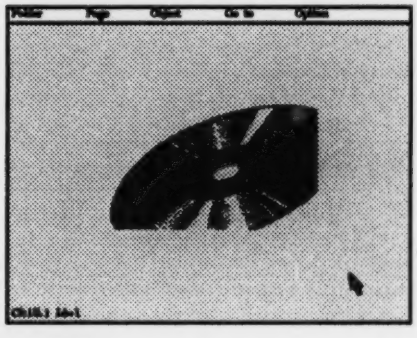


Perhaps you would like to have a "zoom" effect that causes the picture to gradually appear. That is, first you see a blank page, and then the picture appears a little at a time









until the full picture is visible.

First, place your picture on the page. Use a Tracker button to locate the upper left and lower right corner of the picture you are using. Then create a Script button called ClickMe, with a script that will gradually display the picture:

```
object "disc"; /*disc is the name of the picture*/
move 10,5,25,10;
move 10,5,40,15;
move 10,5,55,20;
move 10,4,70,25;
```

When you click on the button, you see the screen change several times as the picture appears. If the picture appears too quickly, you can put **wait** commands between each **move** command. When you are satisfied with the animation, place the picture on the page so that none of it is visible. Then when the button is activated, the picture will appear a "frame" at a time.

This same technique can be used to move a button or field on the screen. If the icon on a button is a small picture, the button can be a vehicle for moving small pictures around the screen. If the object is a field, its size could be changed to reveal more text.

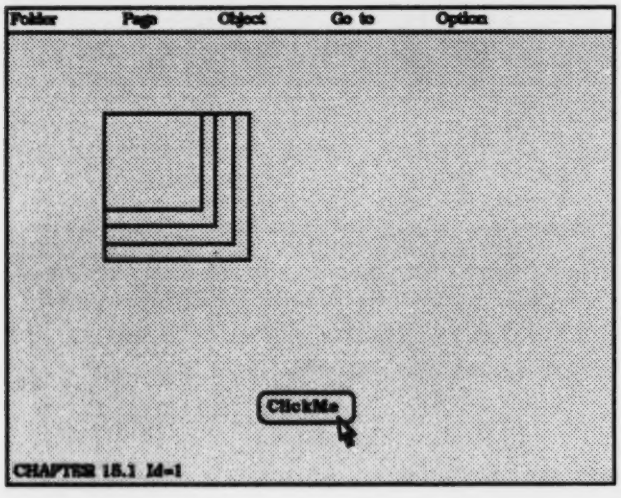
Using Box, Bar, and Line

You can create some interesting effects by using the drawing tools available on the **Page** menu. You can have lines drawn or you can have boxes or bars appear in different colors to emphasize the importance of information on your page.

Modify the script on the ClickMe button as follows:

```
box 1,100,100,200,200;
wait .5
box 1,100,100,210,210;
wait .5
box 1,100,100,220,220;
wait .5
box 1,100,100,230,230;
```

Now click on the button, and you see four squares appear, one at a time.

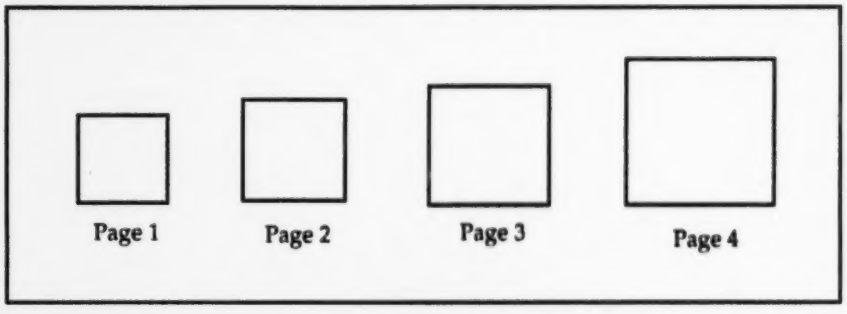


If you then go to the next page and return to this page, the group of squares will be gone.

Flipping Pages

One simple way to create animation is to create a series of pages, each with a slightly different picture on it. Then you can flip through the pages to produce an animation effect, much like you might flip through a flip book in which each page has a slightly different picture.

Create a new folder with four pages. Put a slightly different picture on each page. If you don't have any suitable pictures available, you might use **Box** or **Bar** from the **Page** menu to create a square of a slightly different size on each of the four pages.



If you want the square to be placed accurately, you can use a Tracker button to identify the coordinates of each square.

If you now flip through the pages, you will see the square get larger. You can flip through the pages rather slowly by using the **Go to** menu. Or you can add a Go button on the background that takes you to the next page. Each time you click on the button, the next page of the animation appears. Use one of these techniques to be sure you are satisfied with your animation.

When you are satisfied with your animation, add an "Animate" button to your page. Make this button a Script button and put it on the first page of your animation:

```
@StartLoop
go seq + 1;
if seq < 4 jump StartLoop;
```

The first line of this script is called a *label*. You will learn more about labels in Chapter 24. Note that the label does *not* have a semicolon (;) after it.

The next line uses **seq**. You have seen **seq** previously in this book. **Seq** is a function that returns the *number* of the current page. Its syntax is:

```
seq
```

Note that it does not return the Id number of the page but rather the page number, the number immediately after the name of the folder in the Status line.

The last line of the script uses both **if** and **jump**. You will learn more about **if** in Chapter 21 and more about **jump** in Chapter 24. For the moment you do not need to understand how these commands work. You can simply use this script as a template for such things as animation.

Click on the Animate button. You should see your pictures flip past quickly. You may find that the picture flips so rapidly that you can't see the animation. In that case, you may want to add a **wait** command:

```
@StartLoop
go seq + 1;
wait 0.25;
if seq < 4 jump StartLoop;
```

Note that the input to **wait** is given as a decimal number of seconds. Generally a one-second wait between pages in an animation is too slow. You will probably want to use a decimal after **wait**. Take some time to adjust the timing for your particular machine and preference.

You can use the above loop template for any number of pages that you want to animate. You can start on any page in the folder and flip through the subsequent pages in order. The number in the last line of the script must be the number of the last page of the animation. For example, if you want to stop on page 25 of your folder, the line would be

```
if seq < 25 jump StartLoop;
```

Debugging and Testing

When you are working with animation, it is easy to accidentally destroy the work on your page as you are experimenting. When you have created a page you want to keep, make a copy of it. You can put the copies for safe-keeping at the end of your folder. Then if you make an error, you can delete that page and make another copy of your original to use for your experiments.

It's easy for scripts using animation to get quite long. You should begin by checking a line or two of your animation script before writing a lot of similar commands. If you have a long script and only part of it works, you can use the remark symbols to temporarily remove statements that you know work (or don't work) in order to check other statements. For example, in the following script

```
object "disc";
move 10,5,25,10;
/*move 10,5,40,15;
move 10,5,55,20;*/
move 10,4,70,25;
```

only the first and last **move** commands will run. The middle two will be ignored.

New *LinkWay* Words in This Chapter

Functions
seq

Project Ideas

There are a number of projects that can be developed using animation. Experiment with the different techniques used in this chapter. Which techniques are best for which types of animation? Here are some ideas to try.

- There are many public domain collections of art you can get for use with *LinkWay* (and other graphics programs). Often this is called "clip art." Locate some clip art that interests you and create an animated story. The buttons could be used to cause a variety of actions to occur.
- Create some animated "modern art." You could use a variety of geometric shapes and patterns to create a "painting."
- Create some "art tools" that others could use. That is, you might have a combination of a button and small pictures or icons. These could be copied to another page and used to cause some simple animation.

Chapter 16. Working With the Mouse

You have learned that you can use the mouse to select objects from within scripts using **mouse**. You have also learned that you can find the location of the mouse using **mousex** and **mousey**. In this chapter you will learn how to change the shape of the mouse pointer and how to use the mouse location and the state of the mouse button to cause actions on your page.

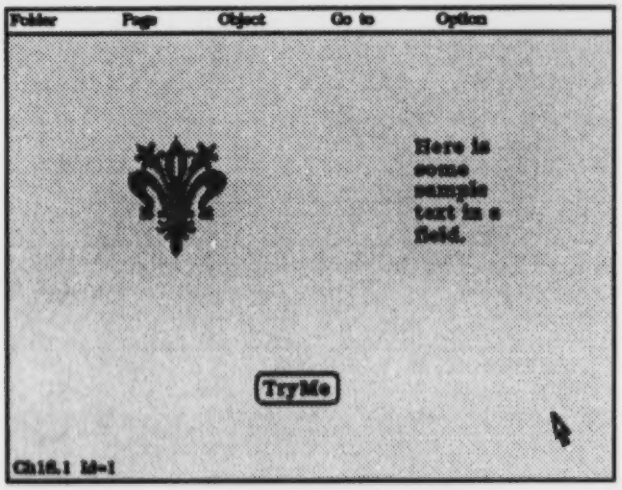
Modifying the Mouse Pointer

You no doubt have noticed that the mouse pointer changes as you move the mouse on the screen. If you move over a button, the pointer changes from an arrow shape to a pointing hand. If you type information into a field, the point becomes a left bracket, [. The shape of the mouse pointer can be changed by using a script.

To begin your work in this chapter, create a new page. Put a Script button called TryMe on the page. Put a sign on the button. Leave the script blank for now, but as you are creating the button, take a look at the Button Icon window to better understand the mouse pointer.

Look for the arrow pointer that you usually see as the mouse pointer in the Button Icon window. You will find a pair of pointers, numbered 75 and 76, that show a white and a black pointer. Looking further you should see a white and black pointing hand, a white and black curved arrow, and a white and black check mark. These pairs of icons are used for the mouse.

Next, create a field and a picture object. Put these on your page as well. Add some text to the field. Your page now looks something like this:



Next, put the following script on the TryMe button:

```
pointer 83, 4, 15;
```

Click on the button. You see the pointer change to a check mark. The inputs to **pointer** are the icon number and the coordinates for the "hot spot," the spot on the icon that is used to initiate actions when you click on the button. Thus the syntax is:

```
pointer icon number, xcoordinate, ycoordinate
```

The standard icons for all modes except CGA, along with the **pointer** command needed to get them, is given in the following table:

Default Arrow	pointer 75,-1,-1
Pointing Hand	pointer 77,4,-1
I-bar	pointer 79,4,4
Curved Arrow	pointer 81,15,6
Check Mark	pointer 83,4,15

With this information, you might expand the previous script so that the icon takes on a different shape depending on what type of object it is over. Recall that **otype** returns the type of the object.

```
mmode 0;
prompt "Click on an object.";
beep;  /*to be sure you have clicked*/
mouse;     /* the button correctly*/
/*I-bar over fields; Curved arrow over pictures*/
if otype = "F" pointer 81,15,6
if otype = "P" pointer 79,4,4;
```

In the previous example, the statement

```
if otype = "F" pointer 81,15,6
```

checks to see if **otype** is the letter "F." If **otype** is "F," the **pointer** command is run. If there is no match, the **pointer** command is ignored and the next **if** command is run. (You will learn more about the **if** command in Chapter 21.)

The **mmode** command is used to force the mouse pointer to remain the shape that you have set. Without the **mmode** command, the mouse returns to the default shapes. The syntax of this command is:

```
mmode pointermode
```

If the pointermode is 1, the mouse shape remains unchanged. If the pointermode is 0, the mouse shape changes as the mouse moves about the screen.

*Working with the mouse pointer can be tricky. If the pointer disappears unexpectedly when you are using **mmode** and **pointer**, check your documentation for the details of **mouse**, low-level control.*

Mouse Buttons and Mouse Location

You have used **mousex** and **mousey** to find the location of the mouse pointer. There is another function related to the mouse, **mouseb**, that returns the state of the mouse buttons. Its syntax is:

```
mouseb
```

The following list shows the values returned by **mouseb**:

0	No buttons down
1	Left button down
2	Right button down
3	Both buttons down

You can use **mouseb** to create the first part of a simple drawing utility. Create a new page. Make a Script button called Line. Place the button at the bottom of the page. This button is going to contain a script that will sense which mouse button is pressed and then draw a line to the left if the left button is pressed and draw a line to the right if the right button is pressed.

The script for this button looks like this:

```
prompt "Click at the starting point.";
mouse;
beep;
if mouseb = 2 line 1, mousex, mousey, mousex+100,
    mousey;
if mouseb = 1 line 1, mousex, mousey, mousex-100,
   mousey;
```

The **prompt** command puts a message in the Status line. The **mouse** command simply waits until a mouse button is pressed. When a mouse button is pressed, there is a beep to confirm that a button has been pressed and **mouseb** returns a 1 if the left button is pressed and 2 if the right button is pressed. A line is drawn 100 pixels to the left of the mouse click if the left button is pressed and 100 pixels to the right if the right button is pressed.

Note that if you are working in Formal Access mode, you must click twice to activate the button, move the mouse, and click once to draw the line.

You can create other buttons to draw boxes and bars. You can create buttons for different sized figures as well. Most likely, you would include instructions somewhere on the page so that the user understands the difference between clicking the left and right button. These instructions might be in a pop-up button or in a field.

Debugging and Testing

Working with the mouse pointer can be tricky. If you have difficulties, remember that a disappearing mouse pointer may have to do with the low-level mouse settings. See your *LinkWay* documentation for details on using these settings.

Did you have problems with the long lines in the last script in this chapter? Remember that long lines sometimes get separated when you edit them. You must use F9 to reconnect lines.

Some of the scripts in this chapter can be rather complex or hard to understand. These are the kinds of scripts in which it is important to include remarks. For example, you should include comments reminding the reader, for example, that a value of 2 from **mouseb** means the right button is down. Taking time to add such clarifying remarks will make your code much easier to work with at a later time.

New *LinkWay* Words in This Chapter

Commands
mmode *pointermode*
pointer *icon number, xcoordinate, ycoordinate*

Project Ideas

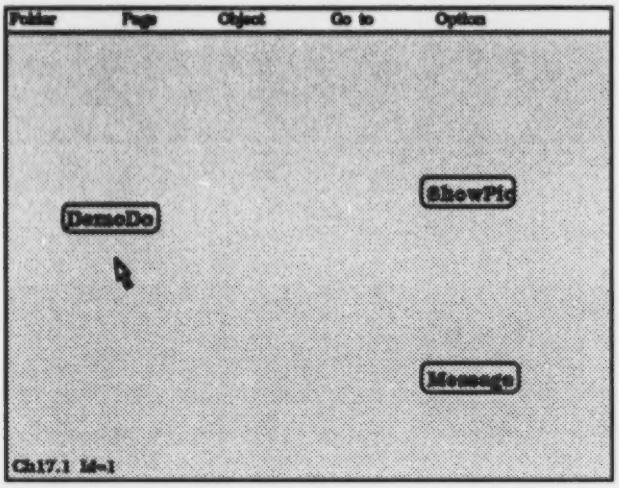
Experiment with changing the shape of the pointer. Try all of the shapes given in this chapter. You might want to create a pointer shape of your own. See your *LinkWay* documentation on **pointer** for more details about how to change the pointer shape.

Further develop the drawing utility example at the end of this chapter. Make buttons to create various sizes of shapes or angles of lines. Can you make a page that really allows the user to make interesting drawings?

Chapter 17. Using Do and List Objects

Previously you saw that you could use Triggers in Reference buttons to activate buttons. In this chapter you will learn how to activate other objects by using a button. You will also learn to create buttons that run automatically when you open a folder.

To begin your work in this chapter, create a page with three buttons. Put a button on the left side of the page called DemoDo. Then create two other buttons. The first button should be a Picture Pop-up button, and the second one should be a Script button that displays a message on the page.



Using the Do Command

The **do** command is used on a Script button to cause an action on some *other* object. The syntax of **do** is:

```
do buttonname
do fieldname
do listobjectname
```

Recall that List objects are available only in LinkWay Live! *If you are using* LinkWay, *you cannot create List objects.*

Before continuing, be sure the two buttons on the right side of the page each work as you had planned. Then put the following script on the DemoDo button:

BEST COPY AVAILABLE

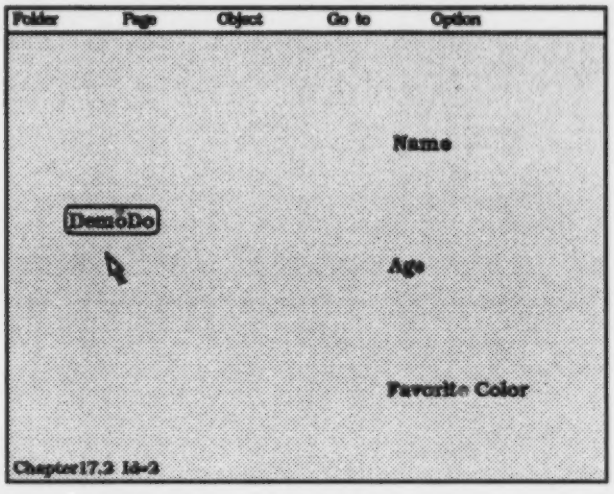
```
do ShowPic;
do Message;
```

Now click on DemoDo. The picture appears as if you had clicked on the ShowPic button. When you click to close the picture displayed by the ShowPic button, the message created by Message is displayed.

Be careful when creating buttons called by a **do** *command. Be sure that every line in the buttons being called ends with a semicolon. If you omit the semicolon, even on a single-line script, you may get an error message.*

You can use the **do** command to have several actions occur automatically in sequence with only one mouse click.

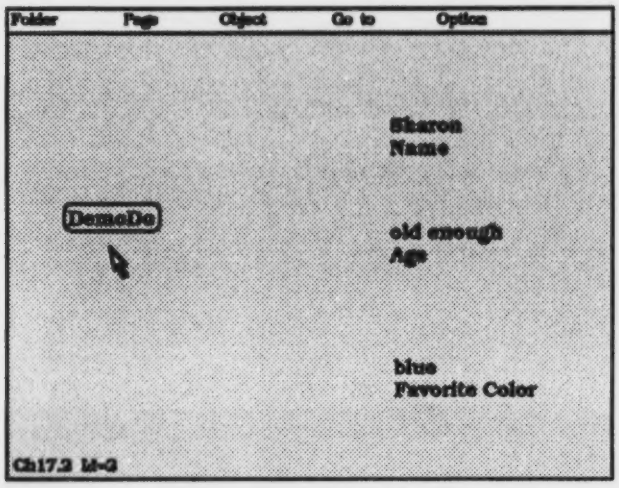
Next create a new page with three fields. Call the top field Name. Put the word "Name" on the second line of the field. Call the second field Age, and put the word "Age" on the second line. Call the third field "Color" and put the words "Favorite Color" on the second line. Cut the DemoDo button from the previous page and paste it onto this new page.



Next, put this script on the DemoDo button:

```
do Name;
do Age;
do Color;
```

Now, when you click on DemoDo, the cursor appears on the first line of the Name field. You can type information on that line. Then you can either press F2 or click outside the field to end the text entry. The cursor next appears on the first line of the Age field, and you can enter text. Again click the mouse or press F2. Finally, you can enter text on the first line of the Color field.



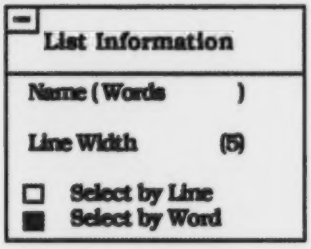
Using **do** on a button is an excellent way to create a page on which the user is to enter data. The script on the DemoDo button guides the order in which the fields are activated.

Note that when you use **do** *to activate a script button, control passes to the button that has been activated. Control does* not *return to the original button with the* **do** *command on it.*

Creating and Using List Objects

If you are using LinkWay, *you cannot use the material in this section. You must be using* LinkWay Live!

List objects are used to allow users to select either a word or a line of text from a window and have *LinkWay* act on the selection. To create a List object, select **New** from the **Object** menu. Place the object on the right side of the page and then select a font of your choice. Name the List object Words and be sure "Select by Word" is checked. Make the line width 5.



List objects allow you to select words or lines of text by clicking on them. You can then use the highlighted information in other scripts.

To get information into a List object, you must use a button. Create a button called Pick. Put the following script on the button:

```
var vwords(30);
set Words = "cat dog rat";
change 1;
```

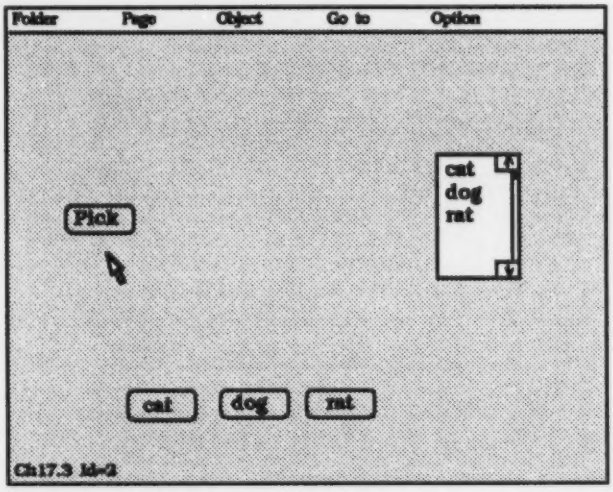
When you click on the Pick button, the words should appear in the List object. The words may not be exactly where you want them. Perhaps you need to make the List object wider. You must use exactly five characters (including spaces) if you want each word on a single line:

```
set Words = "cat  dog  rat  ";
```

Note that each word has two spaces after it in the previous statement.

There are several new commands on this button. The command **var** is used to define a variable. You will learn more about using **var** in Chapter 23. Putting a "v" in front of the name of the List object creates a variable with that name that holds the list of words or the text displayed in the List Object window. Be sure that the number after **vwords** is larger than the total number of characters in the expression after the **set** statement. Finally, **change** is used to redisplay the list object, Words. Note that "change 0" makes a List object invisible.

Now add pop-up buttons to your page for each word in your list. The name of the buttons must correspond to the words in your List object. The text in each pop-up button should describe the animal from the list.



Now, modify the script on Pick as follows:

```
var vwords(30);
set words = "cat dog rat ";
change 1;
object result;
do object;
```

First, click on one of the words in the list Words. Then click on the Pick button. The word clicked on is automatically passed to the function **result**.

Result is a function that receives information from other commands and then passes the values received to yet another command. The syntax of **result** is:

```
result
```

(You will see other examples of the use of **result** in Chapter 21).

If you highlight "cat" and then click on the Pick button, then "object result" means that the button "cat" is selected. Then, "do object" causes the script on "cat" to run, and the pop-up field appears with more information on cats.

When you click on a word in a List object, the values of ***mousex, mousey,*** *and* ***mouseb*** *are also set. See your LinkWay* Live! *documentation under "List objects" for more information.*

Note that if you actually used the previous technique for displaying information, you would probably make the cat, dog, and rat buttons invisible.

Another technique that can be useful for selecting words from a list is the following. Make a script button with no name and no icon. Select Immediate action in the Button Name window. Make the bounding box of the button correspond to the bounding box of the List object. Put the name of the object at the beginning of the script in a **do** command:

```
do Words;
```

When you move the mouse pointer over the List object, control passes immediately from the button to the List object. Next, add the code to select the appropriate button:

```
do Words;
object result;
do object;
```

Now when you click to highlight a word, the highlighted word is put into **result** and the button with the name that matches **result** is run.

There is a great deal of technical information about List objects in your LinkWay Live! *manual. These objects are very flexible and powerful. However, using them requires a fair amount of programming skill. When you have read the last few chapters of this book, try reviewing this section and then read the section on List objects in your* LinkWay Live! *manual.*

Creating and Using AutoExec Buttons

If you are familiar with working with MS-DOS/PC-compatible computers, you have certainly heard the term AutoExec. *LinkWay* also uses this term, but in *LinkWay* you create AutoExec buttons rather than

AutoExec files. AutoExec buttons, as you might expect, run automatically when a page is accessed.

By now you should have several pages in a folder from your work in this chapter. Go to the first page of your folder and create a link button that goes to the third page of your folder. Name the button AutoExec. Now go to the second page of your folder. Return to the first page. The AutoExec button is selected.

Next, change the Access level to the folder to something other than Format. Again go to the second page and back to the first page. When you open the first page, *LinkWay* takes you automatically to the third page. When you are in Format Access level, the AutoExec buttons are highlighted; but in any other Access level, AutoExec buttons are automatically run.

You must return to the Format Access level to modify the AutoExec button. Otherwise, each time you go to the first page, you will automatically be moved to the third page.

Each page in your folder can have *only one* AutoExec button. When you open a page, the AutoExec button runs. The exception is when a script goes to a page but does not end there. An AutoExec button cannot run until the active script finishes.

You can also have an AutoExec button on the base page. If an AutoExec button is on the base page, it is run each time you open any page in the folder.

AutoExec buttons can be used to do initial set-up on a folder or a page. They can be invisible buttons that work without the user being aware that they are there. Remember, however, that you can have only one AutoExec button on a page and that such a button must be named AutoExec.

Debugging and Testing

The material in this chapter is fairly technical and detailed. The more technical the material, the more important it is to use Remark statements. Identify carefully what *any* line of your code does that is not immediately obvious. Clearly identify where objects are found that are used by a **do** command.

When a script doesn't run successfully, take some time to think. Take notes so that you remember what you have tried. Take a break. Sometimes coming back to a problem after a period of time makes the error much easier to find.

Be careful when using **do**. **Do** won't work correctly if the objects you are trying to use are not named correctly. Check carefully for spelling or typing errors. This is especially true when you are using a List object and **result**.

New *LinkWay* Words in This Chapter

Commands
do *buttonname*
do *fieldname*
do *listobjectname*

Project Ideas

Spend some time practicing using **do** to activate buttons and fields. Are you comfortable using one button to activate another? Can you set up a group of fields so that they can be used for data entry?

Expand the "data entry" example from this chapter. Add a different field to hold the words such as "Favorite Color" so that when a field is selected by the **do** statement, the cursor appears to be next to the field-title words.

If you are using *LinkWay Live!*, experiment with List objects. Can you extend the example given in this chapter by changing the words used in the list and modifying the buttons?

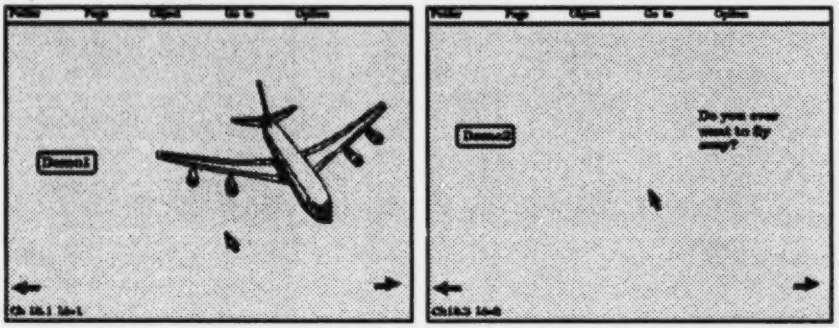
Spend some time exploring the use of AutoExec buttons. Be sure you understand how they work and who you might use them. If you are using *LinkWay*, create a template to use when creating new folders. Create an AutoExec button on the base page that automatically moves to the first page.

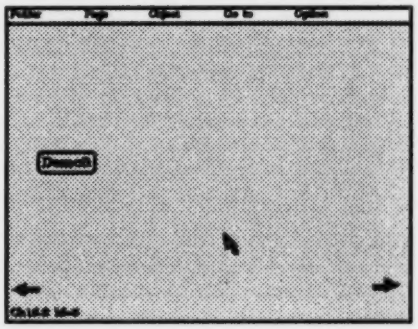
Chapter 18. Working With Pages

You have created new pages from the time you first started working with *LinkWay*. You have also learned to manipulate a variety of objects on a page by using button scripts. In this chapter, you will learn how to use a number of commands and functions to work with pages from button scripts.

You have learned that you can cut, paste, and delete objects from scripts. You have learned that you can move objects from scripts. You have also learned that you can find out information about objects by using a variety of functions. There are also a number of commands and functions that facilitate working with pages from scripts.

To begin your work in this chapter, create a folder with three pages. Put a Script button on each page. Name the buttons Demo1, Demo2, and Demo 3 and leave the scripts blank. On the first page, put a picture. On the second page, put a field with some text in it. Leave the third page blank. You may also want to put Previous and Next buttons on the base page to facilitate navigation among these pages. The specific content of these pages is not important, so don't spent a lot of time creating new material. The pages will simply serve as a vehicle for examining some new commands and functions.





Linking Pages and Folders

Before using a script to link pages, take a moment to examine two functions that give you information about the pages in your folder. Go to the first page in your folder and put this script on the Demo1 button:

```
msg id, count;
```

Id is a function that returns the Id number of the current page. **Count** is a function that returns the number of pages in the folder. When you click on the Demo1 button, you should see a small message window like this:



The first number is the Id number of the current page. The second number is the number of pages in the folder. The syntax of these functions is:

```
id
```

```
count
```

Now put this script on the Demo1 button:

```
link 3;
msg id;
```

When you click on the button, you go directly to the third page in your folder. A small message window appears with the number 3 in it, indicating that you are on the page with Id number 3.

Note that in our simple example, the sequence numbers of the pages and the Id numbers are the same. It is likely that the sequence numbers will

change as you work with a folder. However, the Id numbers always remain the same.

You can also link to another folder with

```
link 5, "More";
```

This command would move to the page with Id 5 in the folder called More. The syntax of the **link** command is:

```
link pageId,[foldername]
```

If the folder name is not included, **link** assumes a link in the current folder.

*Note that you can also use a **go** command to get to the third page. When moving from page to page, it makes no difference whether you **link** or **go**. However, if you want to connect a page with another folder, you must use **link**.*

You can use the **id** function to see if a link has been successful. Modify the script on the Demo1 button as follows:

```
link 4;
if id = 1 msg "No link made.";
```

If you now click on Demo1, you see a message window containing the words "No link made." When a link is unsuccessful, the same page remains on the screen, so you can check to see if the Id number has changed to be sure a link has been properly made.

Adding and Deleting Pages

In addition to using scripts to link pages and folders, you can add and remove pages from your folder by using button scripts. Go to the second page of your folder and put this script on the Demo2 button:

```
clone;
```

Now click on the button. Go to the first page of your folder. Go through your folder one page at a time. You should find that there are now four pages in your folder. The second and third pages are identical. The syntax of the **clone** command is:

```
clone
```

It makes an exact duplicate of the current page.

Next, modify the script on the Demo2 button on the *first* of the two identical pages. The new script should contain only the **delete** command:

```
delete;
```

The syntax of **delete** is:

```
delete
```

Now click on the Demo2 button on the *first* of the two identical pages. Again check the pages in the folder. You now have three pages. Can you tell whether the original second page remains or whether the copied second page remains?

The **clone** command creates a new page and puts it after the current page. After the **clone** command is run, you are on the new page. When the **delete** command is run, the current page is removed and you are on the *next* page of the folder. Thus, in the example just given, after clicking on Demo2 with the **clone** command on it, you are on the new page. When you put **delete** on the *first* of the two identical pages, the first page is removed and the new copy remains. Note that the Id numbers are the keys to help you identify which page you are working on.

These two commands can be used to restore a page that is modified by a script. For example, suppose you want to move the field on the second page. You might use the following series of commands on Demo2 to move the field to the lower left side of the page:

```
object "Example"; /*Example is the name of the field*/
move 35,10,45,18;
wait 0.2;
move 30,13,40,21;
wait 0.2;
move 25,15,35,24;
```

You may need to adjust these commands a bit, depending on the mode in which you are working. When you are satisfied with these commands, modify the script as follows:

```
clone;
object "example"; /*Example is the name of the field*/
move 35,10,45,18;
wait 0.2;
move 30,13,40,21;
wait 0.2;
move 25,15,35,24;
wait 1;
delete;
```

When you click on Demo2, you see the field move. There is a pause caused by the last **wait** command. Then you are on the last page of the folder.

If you check the pages of your folder, you will find that the there are still three pages and that the original second page remains. The modifications were made on the copy of the page and then the copy was deleted. This technique can be very useful for restoring a page to its

original status. However, you must be very careful when using **clone** and **delete**. You can easily destroy your work as you are debugging.

You can also copy and paste entire pages either into the Scrap file or into a file of your choice. If you put

```
cut
```

into a folder, the current page is put in the Scrap file. If you later use

```
paste
```

in a script, the page in the Scrap file replaces the current page on the screen. The syntax of these commands is:

```
cut filename
```

```
paste filename
```

Controlling the Screen

Another technique for restoring the screen without the user being aware of the changes is to use the commands **show, noshow,** and **onshow**.

***Onshow** is available only in* LinkWay Live!

Suppose you want to move a button so that it is on top of a picture, perhaps to draw attention to the picture. Put this script on button Demo1 on the first page of the folder:

```
object "Demo1";
move 40,12,50,14;
wait 1;
move 10,12,20,14;
```

When you click on the button, you see the button move to the right, pause for a second, and then return to the left side of the screen. (You may need to modify the coordinates to suit your working mode.)

Perhaps you don't want the user to see the button move back to the left. If so, you can add a **noshow** to the script:

```
object "Demo1";
move 40,12,50,14;
wait 1;
go 2;
noshow;
go 1;
object "Demo1";
move 10,12,20,14;
go 2;
```

This script moves the button, pauses, goes to the second page, and freezes the screen. The button is then moved on the first page and the script returns to the second page.

You may see a screen flash when using a script like the preceding one. This depends on the speed of the computer you are using.

The screen automatically returns to a normal update mode when a script finishes. You can force a return to the normal update mode with **show** and **onshow**. **Onshow** does *not* cause an immediate update of the current screen; **show** *does* cause an immediate update of the screen.

These three commands all have a simple syntax:

```
show
```

```
onshow
```

```
noshow
```

Another way to control the screen display is the use of the **mbar** and **nombar.** The syntax of these commands is:

```
mbar
```

```
nombar
```

When you include **nombar** in a script, the Menu bar becomes invisible until an **mbar** command is used. You may want to use **nombar** on an AutoExec button on the first page of your folder. This way users won't be distracted by the presence of a Menu bar. When you reach the end of a folder, an **mbar** command can be used. Note that even though the Menu bar is invisible, you can still use it.

Debugging and Testing

When you are working with **clone** and **delete** it is easy to create many extra pages or to erase some of your work. Before you begin including these commands in scripts, you may want to make a copy of your folder for safekeeping. You can select **DOS** from the **Option** menu and type something like

```
copy ch18.lbk ch18.bkp
```

If you accidentally destroy your original folder, you can exit to DOS again and rename your backup file:

```
rename ch18.bkp ch18.lbk
```

Then you can return to *LinkWay* to continue your work. Note that the extension on the file name will vary depending on the mode in which you are working.

When you are using commands like **clone** and **delete**, you can protect your folder by using the **if** command. For example, you might use a statement like this:

```
if id = 5 delete;
```

Such a statement would check to be sure the correct page was being deleted.

Keep in mind that if a **link** command fails, *LinkWay* stays on the same page. If nothing happens after a **link** command, carefully check the Id and folder name of the target page.

Similarly, if there is not enough memory in your computer, a **clone** command has no effect. You can easily check to see if a **clone** command is successful by including

```
if id = 3 msg "No copy made";
```

Here, the page on which the script is placed would have an Id of 3. Instead of a **msg** command, you would probably have some other action occur.

New *LinkWay* Words in This Chapter

Commands
clone
cut *filename*
delete
link
mbar
nombar
noshow
*onshow
paste *filename*
show

Functions
`count`
`id`

*Only available in *LinkWay Live!*

Project Ideas

Experiment with the techniques described in this chapter for restoring pages that change. Which of the two methods works best for which types of changes? Which of the two methods works best on your particular computer?

Spend some time creating a small folder so that it is polished and ready for others to use. Hide the menu bar. Make sure the user only sees appropriate changes on the pages. Make the folder as user-friendly as possible.

Chapter 19. More on Graphics

You have worked with graphics by using *LinkWay Paint*, and you have learned to put pictures from a variety of sources into your folders. In this chapter you will learn some more techniques to use with *LinkWay Paint*. In addition, you will learn to work with Picture Cut buttons.

New Techniques in *LinkWay Paint*

The Shift key is valuable when used in combination with a number of the menu options in *LinkWay Paint*. Open a new picture in *LinkWay Paint*. Select **Box** from the **Draw** menu. If you click the mouse button and then move the mouse, you can draw a rectangle of any shape. However, using the Shift key makes it possible to draw only squares. To draw squares:

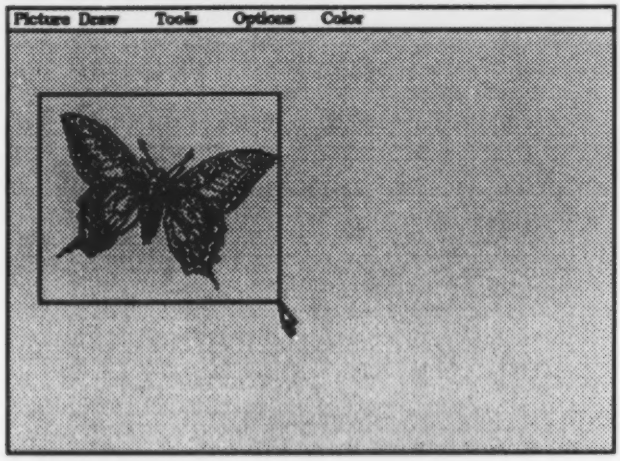
- hold down the Shift key
- click the mouse button
- move the mouse until the square is the size you want
- release the mouse button
- release the Shift key.

Note that you must hold the Shift key down all the time you are drawing the square. This same technique works for **Bar** and **Shad Box.** By using the Shift key, you can draw "filled in" squares with **Bar,** and you can draw square shadowed boxes.

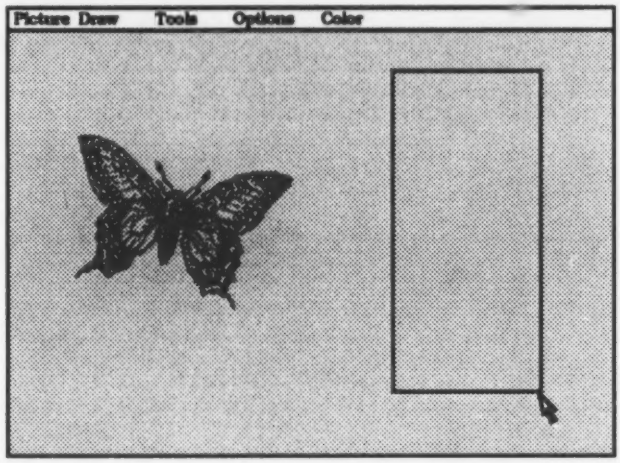
The same technique works with **Circle**. You can draw an ellipse-like shape with the **Circle** command. However, if you hold down the Shift key, you get only circles. To draw circles:

- hold down the Shift key
- click the mouse button
- move the mouse until the circle is the size you want
- release the mouse button
- release the Shift key.

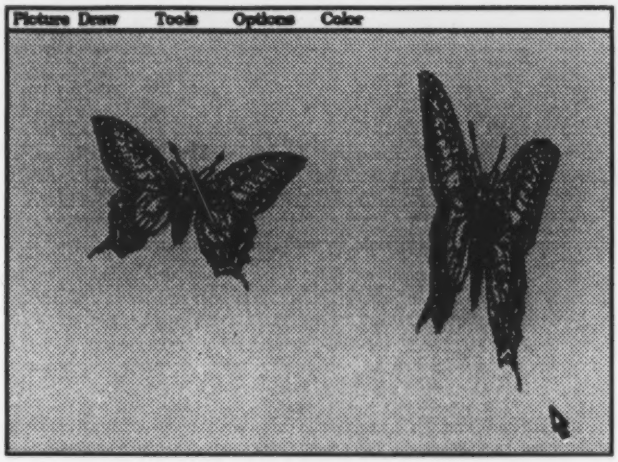
You have probably used **Copy** to select and move parts of a picture. A similar technique works with **Scale**. Choose **Scale** from the **Options** menu. Use the mouse to select a section of a picture that you want to make larger or smaller or of a different shape.



Then draw an area elsewhere on the screen the size and shape you want the revised drawing to be.



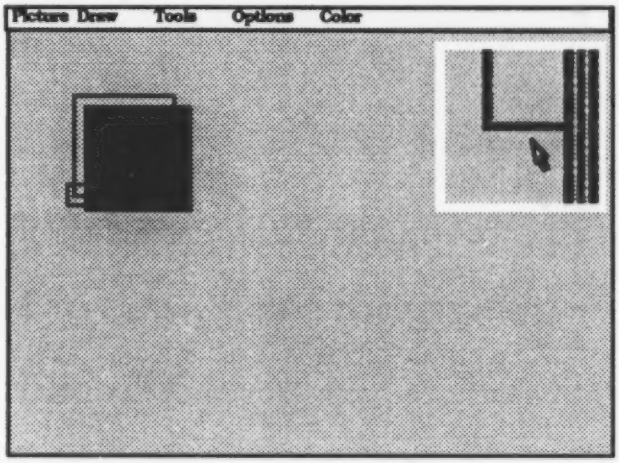
When you release the mouse button, the scaled drawing will appear.



Remember that when you simply copy or scale a picture, the picture covers whatever is underneath it.

The Shift key also adds capabilities when copying and scaling. If you hold down the Shift key when you select **Copy** or **Scale**, the image you select and paste onto the screen is transparent. That is, when the Shift key is used, you can "see through" the parts of the copied image that are the same as the background color.

Magnify is another option that can help you do detail work while working in *LinkWay Paint*. Select **Magnify** from the **Tools** menu. A small box appears. Click over the area you want to work with. A 16-by-16-dot grid appears in the upper right corner of the page:

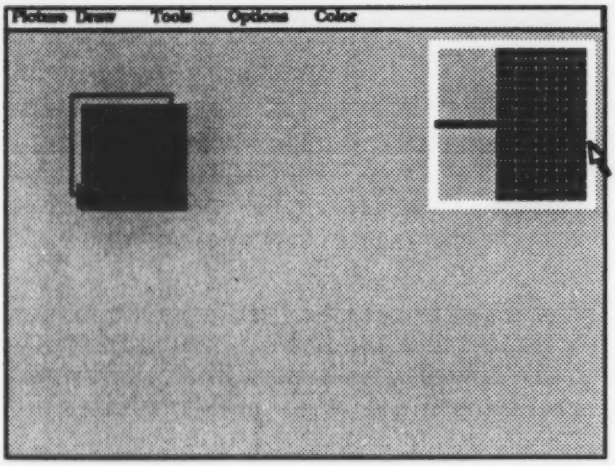


This grid is a magnification of the dots—pixels—in the smaller square you placed on the screen. You can use the grid in the upper right corner to change the pixels on the screen. If you click on a spot in the grid, a dot is

drawn in the current color. If you hold down the mouse button and drag the mouse, every dot touched by the mouse pointer turns to the current color.

There are two ways to change the color of the individual pixels on the screen. You can click on the Color window and then click on a dot to change its color. Or you can hold down the Shift key and click on a pixel in the grid. The color of the pixel you click on becomes the new drawing color. Clicking in the grid now draws in your new color.

You can change the portion of the screen displayed in the Magnify grid. If you click on the border, the part of the screen displayed moves by eight pixels in the direction of that border. For example, clicking on the right border causes the Magnify box to shift to the right eight pixels.



If you are working in color, there are a number of other options that you can use to work with color. Under **Tools** you can select **Fade** and **Smooth**. Under **Options** you can use **Make Fade** and **Mix Color**. Several of these functions depend on the mode in which you are working. See your *LinkWay* documentation for details.

When you are working with colors, you can bypass the **Color** menu by holding down the Shift key and clicking over the color you want to use. The color under the mouse pointer becomes the current color. When you draw, the new color will be used.

The Alt key can be helpful for placing drawings along character grid boundaries. Hold down the Alt key while making a selection from the **Draw** menu. Your drawings will then line up only on character grid boundaries. (You can see these boundaries by selecting **Show Grid** from the **Options** menu.

Picture Cut Buttons

You have made buttons using icons (pictures) and names (signs), and you have probably also made invisible buttons using the None option from the Button Icon window. The fourth option for a button is Picture Cut.

Create a new Script button. Name it Pict, and select Pict Cut from the Button Icon window. You will then see a list of "cut" files that come with *LinkWay*. Select one of the files from the list. A small picture appears in the upper left corner of the bounding box for the button. The name of the button is changed to match the picture you select. Put a script something like this on the button:

```
beep;
beep;
```

Now try clicking on the button. You will notice that the entire bounding box serves as the target area for the button. You should take time to size this button to match the picture cut. Otherwise users may become confused if they click away from the picture and some action occurs.

You can make your own picture cuts to use with your buttons. Often the images used on Picture Cut buttons are similar to icons, but this Picture Cut option provides the flexibility of creating multicolored buttons that specifically suit your needs. Go to *LinkWay Paint* and create a small graphic. Select **Cut** from the **Picture** menu. Carefully select the area of the graphic you want to appear as the picture for your button. When the Cut window appears, type a name to match your *button* rather than using Scrap as you have in the past. Return to *LinkWay*. Now either create a button with the same name as your graphic or modify the existing button so that it has the same name as your graphic. The graphic you created will become the picture for the button. Size the button bounding box to match the picture.

A page of small pictures that can be used for Picture Cut buttons comes with *LinkWay*. There are such pages available for most graphics modes. Using that special page, you may want to create a series of picture cuts for use in future projects.

Debugging and Testing

When you are introduced to new shortcuts like those in this chapter, it is easy to decide that it takes too much effort to learn them. You have gotten along just fine until now without these new tools, and perhaps you feel a bit overwhelmed by all of the new ideas you have learned so far. In spite of such feelings, it is a good idea to take the time to learn skills that will make your work easier. If you have trouble remembering new techniques, make some cards as reminders. Put them near the computer

where you work so that you won't have to look them up in the book. In time you will learn the shortcuts just by practicing them.

Keys Used in This Chapter

Keys	Action
Shift with Copy and Scale	Makes selected graphic transparent
Shift with Box, Bar, Shadow Box	Draws a square box or bar
Shift with Circle	Draw a circle
Shift with Magnify	Picks up color under pointer
Shift with Color selection	Picks up color under pointer
Alt with drawing tools	Forces pointer to character cell boundaries.

Project Ideas

Spend some time learning to use the new keys with *LinkWay Paint*. Be sure you are comfortable with these new tools. Experiment with such things as lining up objects along character grids, changing colors using the Shift key, and drawing perfect squares and circles.

Work with some Picture Cut buttons. Use some of the picture cuts that come with *LinkWay*. Copy some sections from the pictures that come with *LinkWay* to new picture pages and use them on buttons. Design some pictures of your own to use on buttons.

Chapter 20. Interactive Programming

All of the action taken by your *LinkWay* scripts so far has been completely planned by you. In this chapter, you will learn how to accept input from the user and use that input to modify the actions taken by your scripts.

No doubt you have used many computer programs that asked you questions and requested a response from you. Sometimes you are asked to type your name. Sometimes you merely need to click on a button. You can use both of these methods to have a user of your folder interact with *LinkWay*.

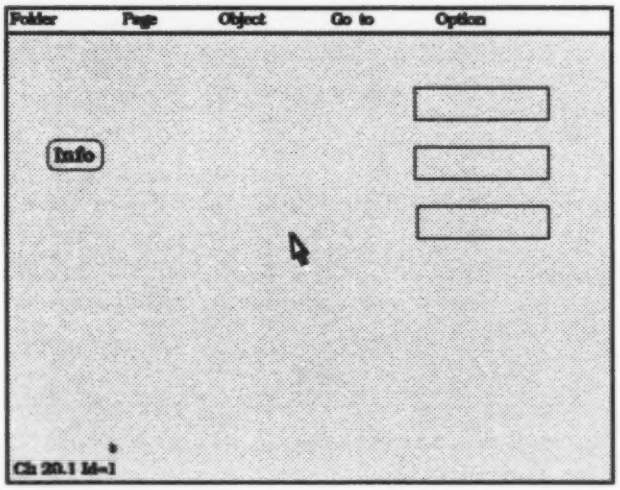
Using Input

The first interactive command is **input**. The syntax of **input** is:

```
input prompt1,[prompt2,]variable
```

Prompt1 and prompt2 are quoted strings that represent the questions to be asked. Variable holds the answer.

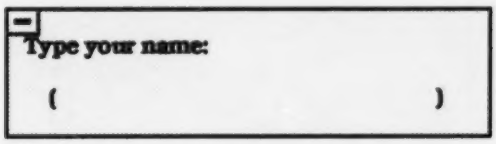
Create a new page to explore the use of **input**. Create three small fields called Field1, Field2, and Field3. You may want to draw boxes around each field so that you can see them, because no text will be put into them right away.



Put a script button called Info on the page and put this script on it:

```
var ans(32);
input "Type your name:",ans;
set Field1 = ans;
```

When you click on the button, you see



If you type your name between the parentheses at the bottom of the window and then click the Close Dot, your name appears in the first field.

Take a moment to examine the preceding script carefully. The first line defines a place for the answer that you type to be stored. This "place" is called a variable. You will learn more about variables in Chapter 23. For now, you can use this first line with any **input** statement. It is set to allow the longest possible answer.

The next line allows you to enter the prompt you want to have at the top of the window created by **input**. The prompt can be no more than 34 characters long. If you use two prompts, each one can be only 34 characters long. The last part of the **input** statement is the name of the variable into which you put the answer.

The final line of the script puts the answer into a field. **Set** was used in this script, but you might want to use **flowset** if the answer is likely to be longer than one line of the field.

Modify the script to accept several answers and put them in different fields:

```
var ans(32);
input "Type your name:",ans;
set Field1 = ans;
input "Type your phone number:",ans;
set Field2 = ans;
input "What state do you live in?:",ans;
set Field3 = ans;
```

If you now click on the Info button, the Input window will appear three times. Each time you enter information, that information is put into the appropriate field. Notice that you can use the variable "ans" several times in the same script. Each time you enter new information, it replaces the previous information stored in "ans."

Previously you learned about the command **drowcol**. You used this command to place the window for a **msg** command. **Drowcol** can also be used to place the window for **input**.

*Recall that **drowcol** can be used only in* LinkWay Live!

Modify the previous script by adding a **drowcol** command:

```
var ans(32);
drowcol 12,5;
input "Type your name:",ans;
set Field1 = ans;
input "Type your phone number:",ans;
set Field2 = ans;
input "What state do you live in?:",ans;
set Field3 = ans;
```

Now click on Info. All three of the windows created by **input** appear on the left side of the page. Like **msg**, the Input window appears in the center of the page unless you use a **drowcol** command. After using the **drowcol** command, the Input windows remain in the same location until a new set of coordinates is entered by using **drowcol**. Note that subsequent **msg** commands will also have windows placed at the same location until another **drowcol** command is used.

Input can also be used to get answers from users that can be checked by using an **if** command. You have seen **if** used before, and you will learn more about **if** in Chapter 21. Modify the script on the Info button as follows:

```
var ans(32);
set Field1 = "";
input "Do you like computers?","yes or no",ans;
if ans = "yes" flowset Field1 = "Good, since you are
     using one!";
if ans = "no" flowset Field1 = "I'm sorry you are
   unhappy!";
```

When you click on the Info button, you are asked to type "yes" or "no." The result displayed in the first field depends on your answer. Note that you must type the answer exactly as shown. If you use, say, capital letters, nothing will appear in the field. Also note that the field is set to "empty" by the command

```
set Field1 = "";
```

You will learn more about checking for a variety of similar answers such as "Yes," "YES," and "yes" in Chapter 21.

Using Menu

The **menu** command lets you create and display menus from which the user can select an answer with the mouse. The syntax of the **menu** command is:

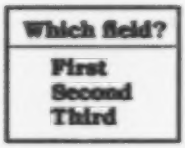
```
menu variablename, width, expression
```

The *variablename* is the place where the result from the **menu** command is placed. The *width* is the number of characters across the menu. The *expression* is the text in the menu.

Using the page from the previous section, modify the script on the Info button as follows. Note that the **menu** command is entered on several lines.

```
var ans(2);
menu ans, 13,
".Which field?":
"-    ":
" First   ":
" Second ":
" Third   ";
if ans = 3 set Field1 = "First field";
if ans = 4 set Field2 = "Second field";
if ans = 5 set Field3 = "Third field";
```

If you click on the Info button, you should see a menu like this:



When you move the mouse over the menu, the lines containing "First," "Second," and "Third" are highlighted. If you click on one of those lines, a message is placed in the appropriate field.

When you try the Info button, you may find that it doesn't work as previously described. Setting up **menu** commands requires that you take a great deal of care. Take a few minutes to analyze the preceding menu command carefully.

The first line of the **menu** statement

```
menu ans, 13,
```

specifies the location of the result of the **menu** command and the width of the menu. The location of the result in this case will be the variable **ans**. Note that when **ans** is defined in

```
var ans(2);
```

it has a "2" after it. That is because the result from a **menu** command can be at most two characters. The second input to **menu** is 13, the maximum number of characters on one line of the menu. *This number must be exactly right or the menu will not be displayed as described.*

The second line of the command

```
".Which field?":
```

displays the text that will appear on the top line of the menu. Notice that the line begins with a period. When a period appears at the beginning of a line, the line becomes nonselectable, and the period is replaced by a space when the menu is displayed. Note that the line ends with a colon. Recall that a colon is used to put together—concatenate—text expressions. In the

syntax for **menu**, note that there is only one expression; thus, the pieces of the expression must be concatenated by using colons.

The third line begins with a dash:

```
"-    ":
```

Any line that begins with a dash is displayed as a solid line in the menu. Note, however, that spaces are put after the dash to fill the line to 13 characters.

Each of the next three lines is similar:

```
" First   ":
" Second  ":
" Third   ";
```

Each one is 13 characters wide. The first two end with a colon to connect them with the following lines. The last line ends with a semicolon because that is the end of the **menu** command.

It is not necessary to set up a **menu** command on multiple lines. However, it is very difficult to get the parts of the menu lined up correctly if you continue to type on the same line. In fact, it probably makes sense to set up the menu by typing each line of the menu on a separate line. Then determine which line is longest. Fill each of the *other* lines with spaces to match the length of the longest line. Your menu will then appear with the entries neatly lined up.

The lines in a menu are numbered by *LinkWay*. The first line, "Which field?," is number 1. The line with the dash is number 2. The subsequent lines are numbered 3, 4, and 5, respectively. When you click on a line, the number corresponding to that line is put into the variable, in this case **ans**. Thus, the **if** statements following the **menu** command in the preceding script check the numbers that might be returned as a result of a mouse click.

Note that there are limits to the size of a menu. A menu can have no more than 16 rows. The menu expression, the part after the width, can be at most 240 characters long. Note also that you can use **menu** to display text that is too long for a **msg** window by putting periods at the beginning of each line so that none of them is selectable. In addition, if you are using *LinkWay Live!* you can use **drowcol** to place your menus.

Menu gives you a way to make your folders quite user-friendly. Users can simply move the mouse and click the mouse button to make selections. They do not have to type information.

Using Fmenu

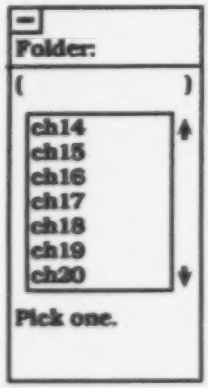
The command **fmenu** is quite similar to the **menu** command. However, instead of providing a list of choices, it displays a file window that allows the user to select a particular file. The syntax of **fmenu** is:

```
fmenu, title, trailer, filespec
```

The *title* is the text that appears at the top of the menu. The *trailer* is the text that appears at the bottom of the menu. The *filespec* is the description of the files that will appear in the window. The statement

```
fmenu "Folders:","Pick one.","*.lbk";
```

causes the following window to appear:



You can then move the mouse up and down the list and click on the file of your choice.

Note that the title and trailer can be only 13 characters long. The file specification can be up to 30 characters long. When you click on a choice in the file list, that choice goes into **result**. The value of **result** is the eight-character file name. The extension is not included. Note also that if you are using *LinkWay Live!* you can use **drowcol** to place a menu created with **fmenu**.

Debugging and Testing

Note that you have reached the point in your work with *LinkWay* that details have become increasingly important. You must be sure that answers typed in response to **input** commands and tested by **if** are exact.

Similarly, working with **menu** can be frustrating. If you don't specify the menu width exactly, the menu will not appear as you expected. Care in the use of periods, dashes, and spaces becomes extremely important if your menus are to look the way you want them to look.

This increasing complexity and need for attention to detail is all part of the process of learning to program. Much of your work with *LinkWay* has been with simple scripts where there were not a lot of places to make errors. From now on, the programming skills you will learn require more care and precise thought. Don't get too frustrated, however. Even

professional programmers spend a large percentage of their time debugging code and testing for possible errors. Even if scripts don't work right the first time—or the second, third, or fourth—you should not be discouraged.

New *LinkWay* Words in This Chapter

Commands
`input`
`fmenu`
`menu`

Project Ideas

Use **input** to create a script that carries on a conversation with the user. Ask the user for his or her name, favorite color, age, favorite music, and so forth. You might use a field to display the answers the user types and your responses to the answers. You might also use a **msg** command to display the responses.

Spend some time exploring the use of **menu** and **fmenu**. You might want to use **menu** to create scripts that ask multiple-choice questions about gender or age. Be sure that you can create menus that work the way you want them to. Try placing menus at different places on the screen. Try menus of different lengths.

Chapter 21. Conditionals

You have seen the **if** command in a number of places previously in this book. In this chapter you will learn more about using the **if** command. In addition you will learn better ways to control the results of the conditions tested in **if** commands.

In your work with **input** you have seen that you can use values entered by the user to cause different actions to occur. Such actions are called *branching*. All computer programming languages include the ability to *branch*. In *LinkWay* scripting, the **if** command is the command that is used for branching. In this chapter you will learn a more flexible way to have *LinkWay* test conditions and respond to the results of those conditions.

Single-Statement If

In its simplest form, the **if** statement is written

```
if condition then statement
```

To begin exploring the use of the simple form of the **if** statement, create a Script button and put the following script on it:

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6 beep;
```

Recall that **ans(2)** allows the value put into **ans** to be two digits. The number entered by the user is compared with 6. If the number is less than 6, the *condition* (**ans** < 6) is **true**; if the number is 6 or greater, the *condition* is **false**. The *LinkWay* code appearing after the **if** and before the statement to be run is called a *condition* and must produce either **true** or **false**.

If you enter different numbers, sometimes you will hear a beep and sometimes you won't. The result depends on the number you type. You can write this script in such a way that the answer typed by the user appears in the Status line:

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
prompt ans;
if ans < 6 beep;
```

Note that in this case the value entered is put into **ans** and then used by **prompt** as well as by **if**.

Multiple-Statement If

Sometimes you want more than one action to occur after you test the result of a condition. For example, you might like to have the beep that occurs be combined with moving to the next page in your folder. To do this, you need the multiple-statement form of **if**:

```
if condition {list of statements}
```

Add a second page to your folder and then modify your script as follows:

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6
{
beep;
go 2;
}
```

Note that you must use the curly braces ({ }) to tell *LinkWay* where the block of statements after **if** begin and end. This form of the **if** structure allows you to have any number of statements after the **if**. Also note that the curly braces in the preceding script are put on separate lines. While this is not necessary, it does help to make complex scripts more readable.

Take a moment to think about how this script works. If the number entered by the user is less than 6, you hear a beep *and* go to the next page. However, if the number picked is 6 or greater, nothing happens. That is, when the condition after **if** produces **false**, all of the statements between the left brace ({) and the right brace (})are ignored.

To examine this idea a bit further, add a statement after the right brace, for example,

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6
{
beep;
go 2;
}
msg "After end of if statement"
```

Now click on the button several times. Enter a number less than 6. *LinkWay* moves to the next page. Enter a number greater than 6. It stays on the same page. However, "After end of if statement" always appears on the page. If the number is less than 6, the statements between the curly braces are run and the message appears on the first page. If the number is 6 or greater, the action between the curly braces is ignored and the message appears on the second page. Be sure you understand what is happening in this example. Take some time to experiment with this new idea.

Single-Statement If-Else

Often when you want branching to occur in your folder, you want one action to occur if the condition is true and another action to occur if the condition is false. In the simple example given previously, you might want the user moved to one page if the number picked were less than 6 and to a different page if the number picked were 6 or greater. Or perhaps you'd like to announce the movement to the next page with a sound if the number were less than 6 or a message if the number were 6 or greater. To accomplish this, you need to use the **if-else** form of the **if** statement. In its simplest form, it is written

```
if condition statement;
else statement
```

Thus, your script might look like this:

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6 beep;
else msg "Going to next page";
go 2;
```

This script moves to the next page in the folder no matter what number is picked because the "go 2" statement is *after* the **if** statement. Because there is only *one* statement after the **if** and *one* statement after the **else**, the **go** statement is not part of the **if** statement.

Multiple-Statement If-Else

There is also a multiple-statement form of the **if-else** structure. It is the most general form of the **if** command:

```
if condition {statement list};
else {statement list}
```

Suppose you want to use the choice made by the user not only to produce different results but also to go to different pages. Your script might look like this:

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6
{
beep;
go 2;
}
else
{
msg "Going to third page";
go 3;
}
```

Add a third page to your folder. Now the user goes to the third page if 6 or greater is picked, or the user goes to the second page if a number less than 6 is picked.

You have seen four forms of the **if** structure. This can be very confusing for a beginner. In fact, it is never necessary to use the single-statement forms of **if** and **if-else**. Even if you have only one statement in each part of the **if**, you can use the multiple-statement form. For example, you can write

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6
{
beep;
}
else
{
go 2;
}
```

instead of

```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if ans < 6 beep;
else go 2;
```

If you find these many forms of **if** confusing, simply use only the multiple-statement format. If you don't need the **else** part, put no commands between the last set of braces.

Using Random Numbers

In the preceding example, you might like to have the computer select a number randomly—much like pulling a number "out of a hat." The function **rnd** will accomplish this for you.

To explore **rnd** a bit, put the following script on a button:

```
msg rnd(10);
```

Click on the button a number of times. You are likely to see different numbers each time you click. You should see numbers in the range from 1 to 9. **Rnd** returns a number between 1 and 1 less than its input. The syntax of **rnd** is:

```
rnd(integer)
```

Thus, you might modify the script from the previous section as follows:

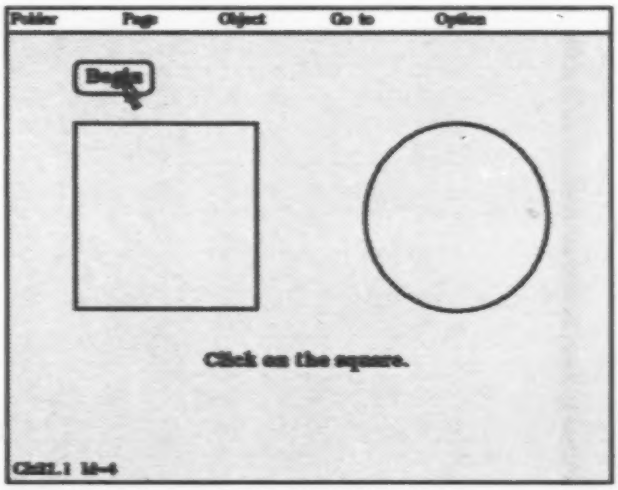
```
var ans(2);
input "Enter a number between 1 and 10:",ans;
if rnd(10) < ans beep;
else go 2;
```

Now the action taken by *LinkWay* depends on whether the number picked by the computer is less than the number entered by the user.

This example is not particularly useful. However, the function **rnd** allows you to build variability into your folders. You can have different responses appear to an action by the user depending on the result produced by **rnd**.

Using the If Structure

The **if** structure gives the programmer a great deal of control over the path taken through the folder. Suppose you have a page that looks like this:



When you click on the Begin button, the message telling the user to click on the square appears. If the user clicks on the square, a "correct" message appears. If the user clicks on the circle, a "You clicked on the circle." message appears. One approach to creating such a page would be to put

transparent Script buttons on each of the two pictures. If the user clicked on the "square" button, a "correct" message would be displayed. If the user clicked on the "circle" button, an "incorrect" message would be displayed.

Another approach to creating this page is to use the **if** command. Recall the **mousex** and **mousey** functions. **Mousey** returns the vertical (up/down) location of the mouse in pixels while **mousex** returns the horizontal (left/right) position of the mouse. Add a field called Answer to the page and put the following script on the Begin button:

```
set answer = "Click on the square.";
mouse;
if mousex < 300
{
beep;
set answer = "That is correct!";
}
else
{
set answer = "You clicked on the circle.";
wait 3;
set answer = "Try again!";
}
```

Note that you will probably have to adjust the 300 in this script to match your screen.

Recall that the **mouse** command waits for a click from the mouse. The user can then move the mouse and click the button. You can then use the values for **mousex** and **mousey** to determine the placement of the mouse on the screen.

If you now click on the square, you see "That is correct!" in the Answer field and you hear a beep sound. If you click on the circle, you see "You clicked on the circle." and then "Try again!" in the field. Spend some time making sure you understand this script. Notice how this uses the multiple-statement form of **if-else**.

Sometimes you have more than two answers to check. Perhaps you have a menu from which to choose. To check multiple responses, you can create a series of **if** commands:

```
if theChoice = choice 1 action 1;
if theChoice = choice 2 action 2;
if theChoice = choice 3 action 3;
if theChoice = choice 4 action 4;
  etc.
```

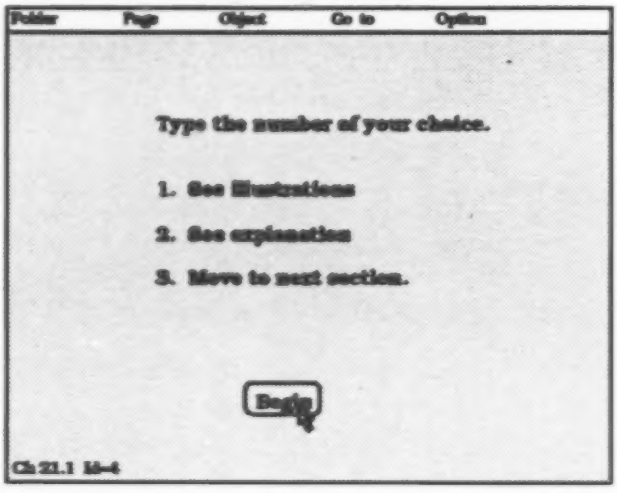
or

```
if theChoice = choice 1
{
action 1;
}
```

```
if theChoice = choice 2
{
action 2;
}
if theChoice = choice 3
{
action 3;
}
if theChoice = choice 4
{
action 4;
}
   etc.
```

In a number of computer languages, this structure is called a "case statement" and is implemented with a case command. In LinkWay *there is no such command. The word "case" has another use in* LinkWay.

Suppose you have a menu with three choices:



Your code might look like this:

```
var ans (2);
set input = "Your choice:", ans
if ans = 1 go 5;
if ans = 2 go 7;
if ans = 3 go 9;
```

If, for example, the user types a 2, the script moves to page 7 of the folder.

Scripts with multiple **if** statements like this can be hard to read. Take time to be sure that your *LinkWay* code is as clear as possible. You can always use comments to separate parts of the code, for example,

```
var ans (2);
set input = "Your choice:", ans
/*See illustrations*/
if ans = 1 go 5;

/*See explanations*/
if ans = 2 go 7;

/*Go to next section*/
if ans = 3 go 9;
```

Using Match

The **match** command is used to compare the user's answer with possible acceptable answers. The syntax of **match** is:

```
match searchstring, expression
```

The search string may be up to 80 characters long, and the expression may be as long as 4,000 characters. Consider the following script:

```
/*Defines the variable ans so that it*/
/*can store up to 80 characters*/
var ans(80);
/*Make sure the field answer is empty*/
set ans = "";
input "What program are you using?",ans;
match ans,"LinkWay,LinkWay Live!";
if result > 0 set answer = "Correct!";
```

The two most likely answers are given in the "expression" part of the **match** commands. **Match** returns a number into the **result** function. If the number is 0, no match has been found. If a match is found, the position in the expression in which the search string is found is returned. That is, if the expression is "kitty cat" and the user enters "cat", the number 7 is put into **result** because the correct match begins at the seventh character. In the previous example, it makes no difference whether you type *LinkWay* or *LinkWay Live!* as an answer. Both answers are accepted.

The two commands **case** and **space** let you further control the effects of a **match** command. The syntax of these commands is:

```
case number
```

```
space number
```

The number after **case** can be either 0 or 1. If it is 0, **match** is not case-sensitive, i.e., "cat" will match "CaT." If the number after **case** is 1, **match**

is case-sensitive, i.e., "cat" will *not* match "CaT." Similarly, if the number after **space** is 0, **match** is not space-sensitive and "cat" will match "c a t". If the number after **space** is 1 it *is* space-sensitive and "cat" will *not* match "c a t."

By adding **case** and **space** commands to the previous script, a wider variety of answers will be accepted as correct:

```
/*Creates the variable answer to accept 80
    characters*/
var ans(80);
/*Makes sure the field answer is empty*/
set answer = "";
input "What program are you using?",ans;
space = 0;
case = 0;
match ans,"LinkWay,LinkWay Live!";
if result > 0 set answer = "Correct!";
```

With the addition of the **case** and **space** commands, a much wider variety of answers will be correct, for example, "linkway", "link Way", and "linkwaylive!"

You can make the range of acceptable answers even broader by using an asterisk (*) in the expression string. The * acts as a "wild card" and matches any string of 0 or more characters. Thus, if the **match** statement were

```
space = 0;
case = 0;
match ans,"l*";
```

any string beginning with the letter "l" would be a correct answer.

***Case and space** also control the way in which **find** commands work. Find buttons and the **find** command are not discussed in this book. See your* LinkWay *documentation for details.*

Debugging and Testing

There are many forms of the **if** statement given in this chapter. You really need to remember only one form:

```
if condition
{
statement list
}
else
{
statement list
}
```

All the other forms are derived from this one.

You can use remarks to help make your scripts easier to read. Notice the last example in this chapter. The remarks remind the reader why the script **ans** has an "80" after it and that setting the field to "" makes sure that the field is empty.

Keep in mind that statements that occur *after* an **if** structure run no matter what happens as a result of the condition after the **if**. That is, in

```
if ans < 6 beep;
else msg "You picked a big number.";
go 7
```

no matter what number is selected, *LinkWay* will move to page 7. If you have problems with this idea, always use the multiple-statement form:

```
if ans < 6
{
beep;
}
else
{
msg "You picked a big number";
}
go 7
```

Here the final brace (}) helps remind you that "go 7" is *after* the **if** structure.

In the Menu example at the end of the section on using **if**, a Begin button was used for asking the user to select from the menu displayed on the page. This is a case where an AutoExec button would be more user-friendly. As soon as the user opens the menu page, the text appears, asking for a choice. The user does not even need to know about clicking on buttons.

New *LinkWay* Words in This Chapter

Commands
`case` *number*
`else`
`if`
`match` *search string, expression*
`space` *number*

Function
`rnd(`*integer*`)`

Project Ideas

Spend some time working with the different forms of the **if** structure. Be sure you understand how each one works. Be sure you understand that commands after an **if** command are always run.

Expand each of the examples given in this chapter. Be sure you understand exactly what the **if** statement is doing in each one.

Make a "quiz" folder in which you check the answers given by students and provide appropriate feedback depending on their answers. Explore the use of **match**, **space**, and **case**.

Chapter 22. *LinkWay* Operators

This chapter examines the rules governing operators that act on or compare constants, variables, or objects. It also looks at the order of precedence of these operators.

In your work in this book, you have occasionally used common mathematical operations, such as addition or subtraction. Even more often you have used an equal sign to compare values. There are a large number of these operators that may be used in *LinkWay*. Much of the material in this chapter is somewhat technical. If you are only using simple arithmetic statements or simple conditions after **if** commands, you may not need much of the material in this chapter. You may wish to refer to this chapter only when you are working with complex mathematical expressions or complex conditions in **if** statements.

Mathematical Operators

As you no doubt know, *LinkWay* allows you to use the standard mathematical operations. Along with addition, subtraction, multiplication, and division, there is a form of division that returns only the remainder of the division. The symbols for these operations are given in the following chart.

+	addition
-	subtraction
*	multiplication
/	division
%	remainder of a division

Examine the following examples to see how the remainder operator works.

```
20 % 3 = 2
```

because 20 divided by 3 gives 6 with a remainder of 2. Similarly

```
5 % 3 = 2
```

because 5 divided by 3 gives 1 with a remainder of 2.

There are also relational operators that are used to compare values or expressions. When the result of a comparison is true, a 1 is returned. When the result of a comparison is false, a 0 is returned. Thus

```
3 = 3
```

returns a 1 while

```
3 = 4
```

returns a 0. The following table gives the numerical relational operators in *LinkWay*.

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
=	equal
!=	not equal

Logical Operators

There are two logical operators, "and" and "or." These operators are used to create more complex **if** statements. The symbols for the operators are given in the following table.

&	and
¦	or

Here are some examples of the use of & and ¦.

```
if ans > 10 ¦ ans < 0 msg "Out of range.";
```

This command checks to see if the value of **ans** is outside the range from 0 to 10. That is, it checks to see if the value of **ans** is greater than 10 *or* less than 0. If either of these conditions it true, it gives an error message. Here is another example:

```
if ans1 = "Yes" & ans2 = "Yes" go 5;
```

This statement checks to see if the first and second answers are both "yes". If so, it moves to the fifth page of the folder.

The use of & and ¦ can become quite complex. It takes practice to develop complex logical expressions. For example, suppose you want to be sure that the mouse is in a certain pixel range. You might write

```
if mousex > 50 & mousex < 100 & mousey > 50 & mousey <
   100 beep;
```

This statement checks to see that the mouse lies in a square 50 pixels on a side that has its upper left corner at (50, 50). You can even check to see if the mouse is in one of two different positions.

```
if mousex > 50 & mousex < 100 & mousey > 50 & mousey <
   100 ¦ mousex > 200 & mousex < 250 & mousey > 150 &
   mousey < 200 beep;
```

Here the mouse would have to be *either* in the square described or in a square that is 50 pixels on a side with its upper left corner at (200, 150).

String Operators

In addition to combining and comparing numbers, you can combine and compare strings in *LinkWay*. Earlier you saw that you can combine strings by using a colon (:), the string concatenation operator. Thus you might have

```
msg "blue":"-":"green";
```

to have "blue-green" appear in a message window. You can use the concatenation operator to combine any number of strings up to a total of 4,000 characters. Often, concatenation is used to combine the value of a variable with other symbols or strings, for example,

```
input "What is your name?",ans;
msg "Glad to meet you, ":ans;
```

This technique allows you to combine the input of the user with other text in a smooth manner.

You can also compare strings. The comparison of strings uses the same operators as the mathematical operators, but with a $ preceding each operator symbol.

$<	less than
$<=	less than or equal
$>	greater than
$>=	greater than or equal
$=	equal
$!=	not equal

When two strings are compared, the first character of the first string is compared with the first character of the second string. Next, the second characters are compared. When two characters that are not equal are reached, the operator returns an appropriate result. For example "catalog" $< "catnip" returns 1, or true, because the second "a" in catalog has a smaller ASCII value than the "n" in catnip. (An ASCII value is a number associated with each letter.)

If the operator is the equal sign and all of the characters are the same, a 1, or true, is returned. Similarly if all of the characters are the same and a not equal is used, a 0, or false, is returned. If two strings are of different

length, the shorter one is treated as if it were filled with spaces to make it the same length as the longer string. In addition, lowercase letters are "bigger" than uppercase letters because their ASCII value is larger.

There is one other way to work with strings. The function **trunc** will remove trailing spaces from a string. The syntax of **trunc** is:

```
trunc(string value);
```

That is, if a string has blanks at the end, **trunc** can be used to remove those blanks. Thus

```
msg trunc("Link "):trunc("Way ");
```

prints "LinkWay" as one word.

Operator Precedence

When several operators are used in the same expression, there are rules that govern which operations are done first. Perhaps you recall learning these "order of operation" rules in a math class in school. For example, if you see

```
3 + 4 * 5
```

the result will be 23 because all multiplications and divisions are done before all additions and subtractions *unless* parentheses are used to modify the expression. The preceding expression would more commonly be written as

```
3 + (4 * 5)
```

for clarity. It is a good idea to use parentheses to make expressions clearer and to insure that computations and comparisons are done in the order you wish. Consider the example used previously in this chapter:

```
if mousex > 50 & mousex < 100 & mousey > 50 & mousey <
   100 ¦ mousex > 200 & mousex < 250 & mousey > 150 &
   mousey < 200 beep;
```

This expression would be clearer if parentheses were used:

```
if ((mousex > 50) & (mousex < 100) & (mousey > 50) &
   (mousey < 100)) ¦ ((mousex > 200) & (mousex < 250)
   & (mousey > 150) & (mousey < 200)) beep;
```

The following table gives the rules for operator precedence. Note that all of the operations at a given level are done on the expression from the left to the right and only *then* are the operations done on the next level.

1	( )
2	* /
3	+ -:
4	< > = <= >= != $< $> $= $<= $>= $!=
5	& |

New *LinkWay* Words in This Chapter

Commands
trunc

Project Ideas

Spend some time experimenting with the mathematical and string operators until you are comfortable with them. Be sure you understand how they work.

Spend some time working with the logical operators. Create if commands using & and | and be sure that they work the way you expect them to work.

Chapter 23. Variables

In your work with *LinkWay* scripting, you have seen variables used in a number of places. In this chapter you will examine the idea of variables in greater depth and learn more about creating your own variables.

Variables

A variable is a container to which you give a name and in which you can store information. Variables can have names that are from one to eight characters in length. The name can contain letters, numbers, and the underscore character (_). Each variable must begin with a letter and may contain no spaces or punctuation. Variables can contain up to 32,700 characters. A given script can contain at most 300 variables.

As you have seen previously in this book, you must use the **var** command to define variables before they are used. The syntax of **var** is:

```
var name(length) [,name(length),...name(length)]
```

That is, you can define as many variables as you want on a single line. Thus, you can write

```
var myName(20);
var myAge(3);
var myPhone(12);
```

or you can write

```
var myName(20);var myAge(3);var myPhone(12);
```

Recall that the number after the variable name is the maximum number of characters that are allowed for the value of the variable.

You have also seen that a value is assigned to a variable using the **set** command. The full syntax of **set** is:

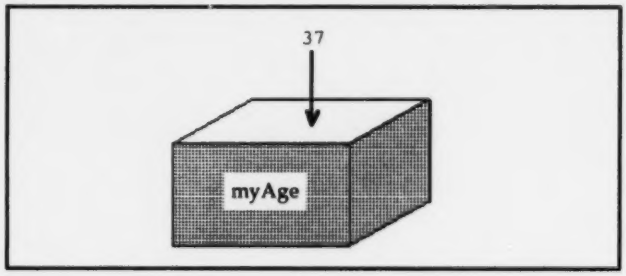
```
set variable = expression
set variable(subscript, length) = expression
set fieldname = expression
set buttonname = expression
set object = expression
```

You have used the last three of these forms of **set** previously in this book. The first two will be discussed in this chapter.

Suppose you want to put the value 37 into the variable **myAge**. You would write

```
set myAge = 37;
```

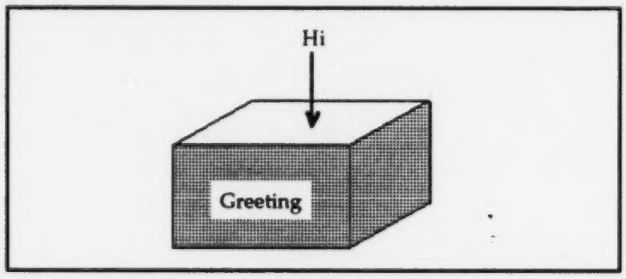
You can think of the number 37 being put into a "box" so that it can be used later:



Or you might type

```
put "Hi" into Greeting
```

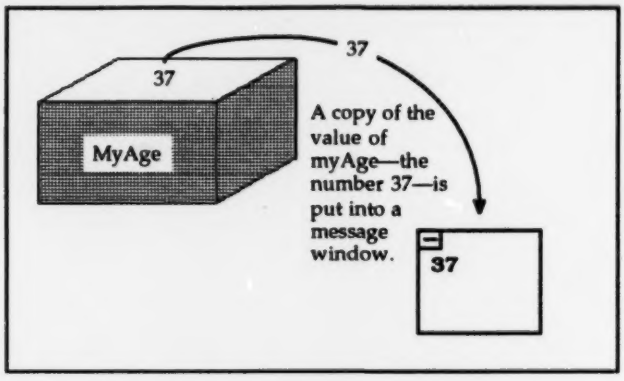
Again, you can think of the string "Hi" as being put into the Greeting box:



In these two examples, "myAge" and "Greeting" are variables. That is, they are named containers that contain values. After typing the preceding commands, you can put in a command such as

```
msg myAge;
```

and the value of **myAge** is put into a message window:



You could also put the value into a field by typing

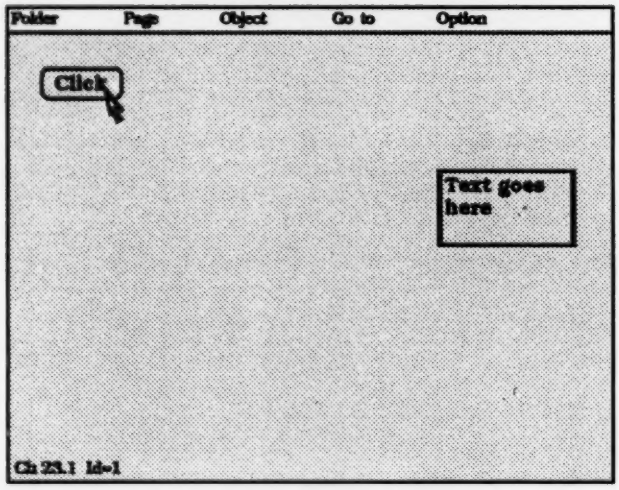
```
set Field1 = myAge;
```

and the number 37 appears in the field named Field1. Note that a *copy* of the value of the variable **myAge** is put into the field. **MyAge** still has the value 37. Thus "myAge" is a sort of container holding a value that can then be put elsewhere at a later time.

Using Variables

Suppose you want to allow the user to decide where some information is to be placed on the screen. You want to draw a box and place a field inside that box wherever the user specifies.

The following example works only in LinkWay Live! *because the command* **wmove** *is not available in* LinkWay. *If you are using* LinkWay, *you may want to develop a similar example using* **move**.



You might begin by creating a script button something like this:

```
/*Define variables to store the mouse location.*/
var xcor(3),ycor(3);
msg "Click where you want to place the field.";
/*Wait for a mouse click.*/
mouse;
/*Put the mouse location into the variables*/
set xcor = mousex;
set ycor = mousey;
/*Draw a box*/
box 1, xcor, ycor, xcor + 100, ycor + 50;
/*Select the field Info*/
object "Info";
/*Move the field inside the box*/
wmove pxwx(xcor),pyxy(ycor),pxwx(xcor + 100),pywy(ycor
   + 50);
```

First, the variables are defined by using the **var** command. Each variable is allowed to be up to three characters long. Three characters is enough because no coordinate is ever longer than three characters. Next, a message is put on the screen, and **mouse** waits until the mouse button is clicked. The location of the mouse at the time the button is clicked is given by **mousex** and **mousey**. Because we want to use these values several times, they are put into the variables **xcor** and **ycor,** respectively.

Next, the coordinates are used to draw a box on the screen. Since **mousex** and **mousey** return coordinates in pixels and **box** needs pixel coordinates to draw, **xcor** and **ycor** can be used to draw a box that is 100 pixels wide and 5 pixels high.

Finally, the field needs to be moved inside of the box. The command **wmove** requires the use of world coordinates, but **xcor** and **ycor** contain values given in pixels. The functions **pxwx** and **pywy** convert world coordinates to pixel coordinates. Thus

```
pxwx(xcor)
```

returns the world coordinates that correspond to the pixel coordinates of **xcor**. Examine the **wmove** command carefully to see how the coordinates for placing the field are computed.

*The functions **pxwx, pywy, wxpx, and wypy** are available only in* LinkWay Live!

The syntax of the functions to convert pixel coordinates to world coordinates is:

```
pxwx(pixelxcoordinate)
pywy(pixelycoordinate)
```

There are also functions that convert from world coordinates to pixel coordinates:

```
wxpx(worldxcoordinate)
wypy(worldycoordinate)
```

Note that there are two other kinds of variables available in *LinkWay*. The type of variables called "pointer variables" will not be discussed in this book. See your *LinkWay* documentation under "Variables" for details. The other type of variable is called a *parameter*. Parameters will be discussed in Chapter 25.

Blanks and Subscripts in Variables

If you define a variable to have 10 characters, and then put a value in the variable that is less than 10 characters, the variable is filled with blanks. That is, in

```
var FirstNm(10);
set FirstNm = "Sharon";
```

the actual value of **FirstNm** is

```
Sharon_ _ _ _
```

where the underline characters represent a blank space. You can check the length of the value of a variable with the **vlen** function:

```
vlen(variable)
```

Thus, you might type

```
var FirstNm(10);
set FirstNm = "Sharon";
/*See the length of the variable*/
msg vlen(FirstNm);
```

You see a message box with the number 10 in it, even though "Sharon" is only six characters long. You can demonstrate that the word "Sharon" is padded by blanks by concatenating a symbol before and after the value of the variable:

```
var FirstNm(10);
set FirstNm = "Sharon";
/*Demonstrate that the variable has 10 characters*/
msg "*":FirstNm:"*";
```

You see

```
*Sharon *
```

in the message window.

You can remove these excess trailing blanks with the **trunc** function, which was discussed in the previous chapter.

For example, if you type

```
var FirstNm(10);
set FirstNm = "Sharon";
/*Demonstrate that trunc removes blanks*/
msg "*":trunc(FirstNm):"*";
```

you now see

```
*Sharon*
```

in the message window.

When you are defining variables you should think carefully about how much space is needed by the variables. You want to be sure that you allow enough space for the maximum characters you need. If too many characters are entered for a variable, the last characters are ignored. However, you don't want to give too many characters to your variables because they consume space in memory.

You can also reference part of the value of a variable by using subscripts. That is, if you have the variable FirstNm and you want to print the second, third, and fourth characters, you could write

```
msg FirstNm(2,3);
```

The first number in the parentheses after the name is the number of the character at which to begin counting; the second number is the number of characters. If FirstNm contained "Sharon" you would see "har" in the message window.

Suppose you are writing a script to create a user name or a password. You might ask the user for his or her name and then generate a combination of characters to use for a password or username from that name:

```
var FirstNm(10),LastNm(10);
/*Get the first name*/
input "First name:",FirstNm;
/*Get the last name*/
input "Last name:",LastNm;
/*Create and print the user name*/
msg "Your username is: ":LastNm(1,5):FirstNm(1,3);
```

The username is made up of the first five characters of the last name and the first three characters of the first name. That is, if the first name is "Sharon" and the last name is "Yoder" the username would be YoderSha.

The ability to access parts of the values of variables can be a very powerful tool, especially when you want to use **if** or **match** to compare answers.

Scope of Variables

When you create a variable in a script such as we have just done, it is *global*. That is, its value is available anywhere in the script, folder, or any linked folder. In a script used previously in this chapter, the coordinates of

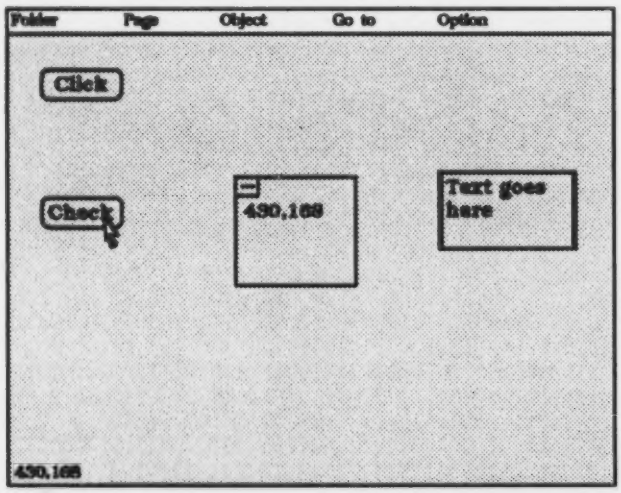
the mouse were used to place a box and a field. Add a line to that script to allow you to see the coordinates of the mouse:

```
/*Define variables to store the mouse location.*/
var xcor(3),ycor(3);
/*Wait for a mouse click.*/
mouse;
msg "Click where you want to place the field.";
/*Put the mouse location into the variables*/
set xcor = mousex;
set ycor = mousey;
prompt xcor:",",ycor;
/*Draw a box*/
box 1, xcor, ycor, xcor + 100, ycor + 50;
/*Select the field Info*/
object "Info";
/*Move the field inside the box*/
wmove pxwx(xcor),pyxy(ycor),pxwx(xcor + 100),pywy(ycor
   + 50);
```

Next, add another button to the page that has this script on it:

```
msg xcor:",",ycor;
```

Now click on the first button and then click on the second button. You see something like this:



When you click on the Click button, the coordinates appear at the bottom of the screen. When you click on the Check button, the *same* coordinates appear in a message window even though the mouse has moved.

Clearly the values of **xcor** and **ycor** that are created when you activate the Click button are still available when you activate the Check button. Try clicking in different places. You will see that the values change each time *and* that they change in *both* places.

You can get rid of variables you no longer want by using the **vclear** command:

```
vclear
```

This removes all of the variables from memory. If you want to release most of the memory taken up by a single variable, you can reset its length to zero with a **var** statement. For example

```
var FirstNm(0);
```

releases the memory used by the variable FirstNm.

Any variables created in a script, such as those just given, are global. Previously in this book (Chapter 1) you learned about **proc** and how to create procedures. Any variables that are defined within a procedure are local. That is, after the procedure is finished, those variables no longer exist. Suppose you rewrite the preceding script as follows:

```
/*Define variables to store the mouse location.*/
var xcor(3),ycor(3);
call Pick;
call BoxFld;
```

Then create a Pick button:

```
proc;
msg "Click where you want to place the field.";
/*Put the mouse location into the variables*/
set xcor = mousex;
set ycor = mousey;
return;
```

Next, create a BoxFld button:

```
proc;
var Name(10);
/*Draw a box*/
box 1, xcor, ycor, xcor + 100, ycor + 50;
/*Select the field Info*/
object "Info";
/*Move the field inside the box*/
wmove pxwx(xcor),pyxy(ycor),pxwx(xcor + 100),pywy(ycor
     + 50);
input "Enter your name: ",Name;
set Info = Name;
return;
```

Notice that in the procedure BoxFld, the variable Name is defined and used to put the user's name in the field. Name is a *local* variable. When the **return** on BoxFld button is reached, Name no longer exists. You cannot use the value of Name created in BoxFld in the main part of the script or anywhere else in the folder.

Debugging and Testing

It is easy to forget that variables are global. That is, you can easily use the same name in different places and accidentally redefine the length of the variable. For example, you might define

```
var Name(30);
```

to be used on one button for the user's full name. However, later in the folder, you might accidentally redefine Name to be the first name of the user:

```
var Name(10);
```

This can cause confusion when shortened full names begin to appear. The problem with variables changing values unexpectedly is called a *side effect* and is easily caused when all variables are global. One solution to this problem is to learn to use parameters (see Chapter 25) and to use local variables in procedures.

Keeping track of all your variables can be a major problem in a large folder. It might be a good idea to create a separate page containing a large field. Each time you define a new variable, list it there along with its size and where it is used. You are much less likely to have problems with side effects if you do this.

Always remember that variable values are padded with blanks. You must take these blanks into consideration when printing the values of variables. Be sure to remember to use **trunc** to get rid of trailing blanks.

New *LinkWay* Words in This Chapter

Commands
`var name(length),[name(length),...name(length)]`
`vclear`

Functions
`*pxwx(pixelxcoordinate)`
`*pywy(pixelycoordinate)`
`trunc(variable)`
`vlen (variable)`
`*wxpx(worldxcoordinate)`
`*wypy(worldycoordinate)`

*Available only in *LinkWay Live!*

Project Ideas

Spend some time experimenting with variables. Put values into variables. Put the values into a message window. Put the values into fields. Experiment until you are comfortable with this new idea. Are you sure you understand the difference between global and local variables?

Explore the use of subscripts with variables. Practice truncating blanks from the ends of variable values. Make sure you are comfortable with these ideas.

Have you tried to write scripts before that really needed variables? If so, use this new idea to try rewriting some of them.

Chapter 24. Looping

Previously you learned to use labels and the **jump** command to create animation. You saw that you could use this structure to do a series of actions a particular number of times. Repeating the same action over and over is called *looping*. In this chapter you will learn a number of other ways to use loops.

In Chapter 21 you learned about branching. You learned that you could control the path taken through your folder based on the result of some condition. Branching is an important component of any programming language. Another component of all programming languages is *looping*. Looping allows a programming language to repeat a task over and over. You have seen that in *LinkWay* you can create a loop by using labels and **jump**.

Looping a Specific Number of Times

In Chapter 15 you learned to use **jump** and **seq** to flip through your folder a specific number of times. Recall that **seq** is a function that returns the number of the current page so that adding 1 to **seq** makes it easy to go to the next page. You can use a variable to move through a number of pages to create animation.

Suppose you have a folder with a Script button named Animation on the first page. On each page there is a NextPg button of type Go that links to the next page of the folder. Suppose there are 10 pages in the folder that contain pictures to create an animation:

```
var counter(2);
/*initialize the counter*/
set counter = 1;
@Startloop
/*Run the button Next Pg that goes to the next*/
/*page in the folder.*/
do NextPg;
wait 0.5;
set counter = counter + 1;
if counter < 10 jump Startloop;
```

When you click on the Animate button, the loop begins. The NextPg button on the first page is run and the second page is displayed. Then the NextPg button on the second page is run. This continues through all 10 pages. **Counter** keeps track of the number of pages that have been "visited."

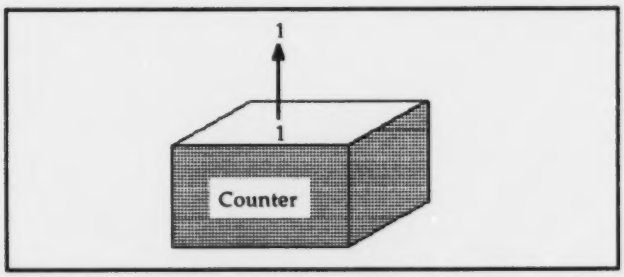
The labels used to create loops are made of letter characters and numbers and end with a character that is *not* a number or a letter. You do not put a semicolon after a label, and each label must begin with an @. The **jump** command is used go to the label. Its syntax is:

```
jump labelname
```

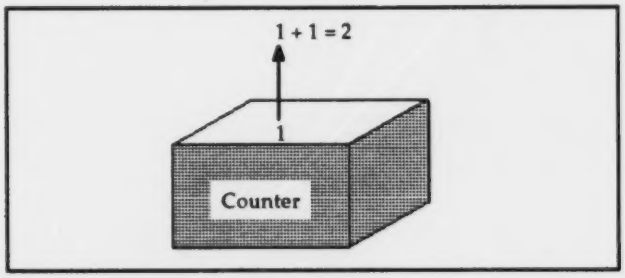
In the preceding example, the variable **counter** is used to control the loop. The value of the variable is set to 1 before the loop begins. Then, 1 is added to **counter** each time the loop is run. The

```
set counter = counter + 1;
```

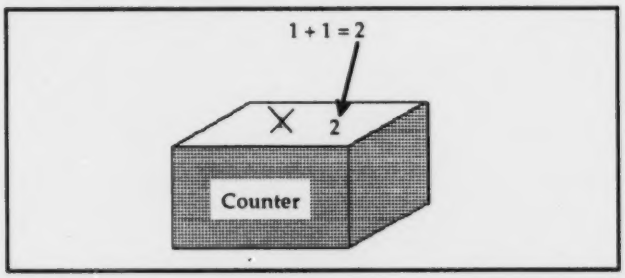
statement accomplishes this task. Recall that variables are like boxes into which values can be placed. This statement takes out a *copy* of the value of **counter,**



adds 1 to it,



and replaces the old value with a new value:



Each time through the loop, 1 is added until the value of **counter** becomes 10.

You can write this same loop with the use of a **stop** command. **Stop** simply causes a script to cease running. Its syntax is:

```
stop
```

The preceding script could be written

```
var counter(2);
/*initialize the counter*/
set counter = 1;
@Startloop
if counter > 10 stop;
/*Run the button Next Pg that goes to the next*/
/*page in the folder.*/
do NextPg;
wait 0.5;
set counter = counter + 1;
jump Startloop;
```

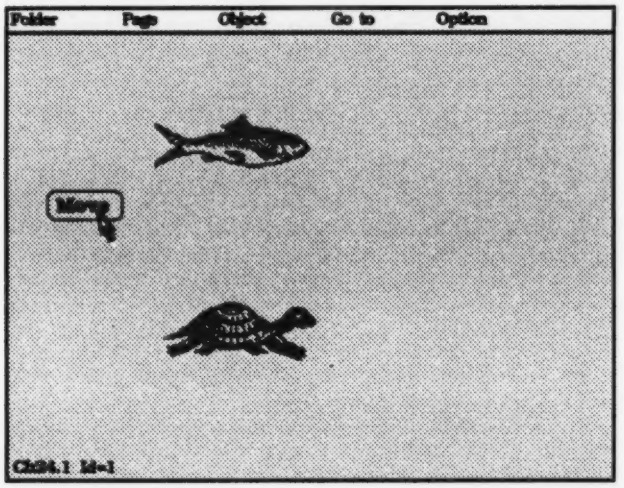
This type of loop is a "test before" loop. That is, there is a test at the beginning of the loop to determine if the loop will be run. The previous script is a "test after" loop.

Sometimes a test-before loop is more appropriate, and sometimes a test-after loop is more appropriate. A test-before loop makes it possible for a loop to be skipped entirely under the right conditions. A test-after loop assures the programmer that the loop will be run at least once. In the preceding script, either type of loop works equally well.

Testing a Condition to Exit a Loop

In the preceding example, there was a particular number of pages that contained graphics to create an animation. There are many circumstances like the one just described in which you want to use loops to complete a task a particular number of times. However, it is also common to have a task in which the number of times you want to cycle through a loop is not exact.

Suppose you have a page containing two pictures. You want the user to click on one of the pictures to have it "fly" across the page.



When users click on the Move button, they are asked to click on a picture. The picture then moves to the right side of the page. The following script will accomplish this:

```
/*define variables to store the left and right*/
/*edges of the object*/
var left(3), right(3);
msg "Click on the object to move.";
mouse;
/*Get the left and right coordinates*/
set left = oleft;
set right = oright;
@BeginLoop
if right > 75 stop;
/*Use functions for the top and bottom coordinates*/
move left, otop, right, obot;
/*Increment the left and right coordinates*/
set left = left + 2;
set right = right + 2;
jump BeginLoop;
```

The loop is a test-before loop. If the object is already at the right edge of the screen, nothing happens. Otherwise, the object moves two character spaces to the right each time through the loop.

If you are using a graphics mode with fewer character spaces across the screen, you will need to make the 75 in the preceding script much smaller.

Note that the preceding loop will work for any object on the page. It will move an object from any place on the screen to the right of its "current" position. This general form of a loop is more flexible that the one in the first section of this chapter that repeats only a specific number of times.

Modularizing Your Scripts

As your scripts get longer and more complex, it becomes increasingly important to take time to break them into smaller pieces. Suppose you expand on the preceding script so that it puts some information into a field depending on which picture is picked and then moves the picture across the page. The script might look like this:

```
/*define variables to store the left and right*/
/*edges of the object*/
var left(3), right(3);
msg "Click on the object to move.";
mouse;
/*Get the left and right coordinates*/
set left = oleft;
set right = oright;
/*Display an appropriate message*/
if oname = "Turtle"
{
set InfoFld = "Watch the turtle go!";
}
else
{
set InfoFld = "Watch the fish swim!";
@BeginLoop
if right > 75 stop;
/*Use functions for the top and bottom coordinates*/
move left, otop, right, obot;
/*Increment the left and right coordinates*/
set left = left + 2;
set right = right + 2;
jump BeginLoop;
```

You can rewrite the script as follows:

```
/*define variables to store the left and right*/
/*edges of the object*/
var left(3), right(3);
call getObj;
call showInfo;
call moveObj;
```

You then need to create three button. First create getObj:

```
proc;
msg "Click on the object to move.";
mouse;
/*Get the left and right coordinates*/
set left = oleft;
set right = oright;
return;
```

This next procedure would go on a button called showInfo:

```
proc;
/*Display an appropriate message*/
if oname = "Turtle"
{
set InfoFld = "Watch the turtle go!";
}
else
{
set InfoFld = "Watch the fish swim!";
}
return;
```

The final procedure goes on a button called moveObj:

```
proc;
@BeginLoop
if right > 75 stop;
/*Use functions for the top and bottom coordinates*/
move left, otop, right, obot;
/*Increment the left and right coordinates*/
set left = left + 2;
set right = right + 2;
jump BeginLoop;
return;
```

It is much easier for you and for other people to understand your code if it is broken into smaller parts. You will learn more about using procedures in the next chapter.

New *LinkWay* Words in This Chapter

Commands
stop

Project Ideas

Spend some time working through the examples given in this chapter. Enhance them. Take plenty of time to be sure you understand how each of these different types of loops works.

Create an animation using several pages. Try using different forms of the loops to display your animation.

Expand on the example in this chapter in which the user clicks on pictures. How can you get the pictures back to the original spot so the next user of the folder can see the page the way it was originally designed? What other features might you add to this example?

Chapter 25. Using Parameters

You have learned how to create variable names and use them to store values. These variables can be used to access values in a variety of different scripts and from a variety of objects in your folder. In this chapter you will learn a different way to move values from one script to another.

Previously you learned a number of things about variables. You learned that:

- You can create variables whenever you need them.
- You can use up to eight letters, digits, and/or underline characters for the name of a variable.
- Values of the variables defined in a "main" script can be used anywhere in folder or in folders linked to the folder having the variable definition.
- Variables that are understood everywhere are called *global variables.*
- Any variable defined in a procedure is *local* and ceases to exist after the procedure runs.

In this chapter you will learn about *parameters*. There are many similarities between variables and parameters. The rules for naming variables also apply to naming parameters. Parameters are used to hold values. However, parameters are *local* to the script in which they are defined, yet they can be used to pass values among scripts.

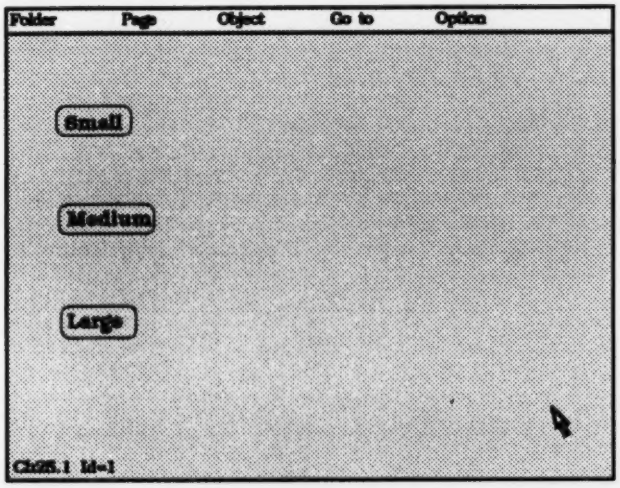
You have used *LinkWay* commands that require parameters, for example, the command **move**. If you just enter

```
move
```

into a script, you get an error message. You must use four values as inputs to **move**. These inputs are values that go into the parameters representing the left, top, right, and bottom coordinates of the selected object.

Getting Started With Parameters

Suppose you want to draw boxes of three different sizes using different buttons. You might have a page that looks like this:



To draw the boxes, you could put the following script on the Small button:

```
box 1,200,100,250,150;
```

Then you would need to put a similar script on the Medium button:

```
box 1,200,100,300,200;
```

Finally, you would put yet another similar script on the Large button:

```
box 1,200,100,350,250;
```

Look carefully at these simple scripts. In what ways are they different? In what ways are they the same? The only difference among these scripts is the number added to the last two inputs to **box**. When you have very similar scripts that differ only slightly from each other, using parameters will probably be useful.

In this case, you might write the procedure

```
proc (Bottom, Right);
box 1, 200, 100, Bottom, Right;
return;
```

Recall that the syntax of **proc** is:

```
proc [(value, value,…value)]
```

Put the preceding procedure on a button named MakeBox. To use this procedure, you need to use **call**. Recall that the syntax of **call** is:

```
call scriptButtonName [(value, value…value)]
```

Put

```
call MakeBox(250,150);
```

on the Small button. Put

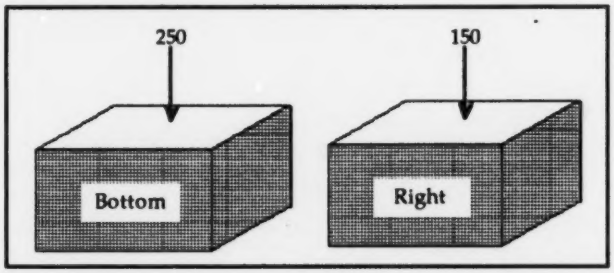
```
call MakeBox(300,200);
```

on the Medium button. Put

```
call MakeBox(350,250);
```

on the Large button.

In this MakeBox procedure, **Bottom** and **Right** are *parameters*. They accept the values passed into them by the **call** statements on the Small, Medium, and Large buttons. When the procedure MakeBox is finished running, the values of **Bottom** and **Right** disappear. Space for **Bottom** and **Right** are automatically allocated as values are passed into them from the main script. If you click on the Small button, two values are passed to the procedure in order to draw a small square:

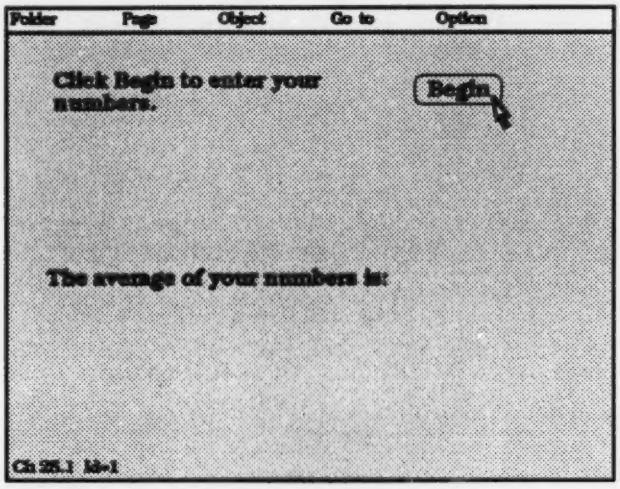


The MakeBox button can be anywhere in your folder. It could be located on an invisible Script button on the current page. However, it is more common to have a page or two at the beginning of the folder just to store scripts used throughout the folder.

Parameter passing is a fairly sophisticated idea. If your only programming experience is with *LinkWay* or an older version of BASIC, this is a completely new concept for you. Give yourself time. Don't expect to completely understand what is happening immediately. As you develop your *LinkWay* scripting skills, you will begin to see where parameter passing would be of value.

Returning a Value From a Procedure

After you are comfortable with the preceding example, consider this next example. Suppose you want to create a page that will allow you to find the average of a series of numbers entered by the user. Your page might look like this:



The script on the Begin button might look like this:

```
var counter(4),number(5),sum(7),average(4);
/*Initialize variables*/
set counter = 1;
set average = 0;
set sum = 0;
/* */
input "Enter a number (0 to stop):",number;
/* */
@AddLoop
if number = 0 stop;
set sum = sum + number;
set counter = counter + 1;
input "Enter a number (0 to stop):",number;
jump AddLoop;
/* */
call average(sum, counter);
set AnsFld = result;
```

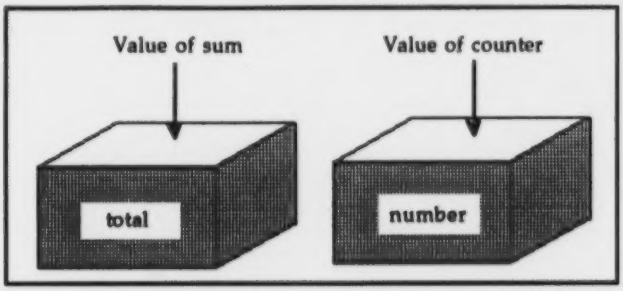
On a script button called Average, put this procedure:

```
proc (total, number);
var ave(5);
set ave = total / number;
return ave;
```

Recall that the syntax of **return** is:

```
return [value]
```

When the script is run, the value of the variable **sum** is put into the parameter **total**, and the value of the variable **counter** is put into the parameter **number**.



The procedure on the Average button computes the average and puts it into the *local* variable **ave. Ave** is local because it is defined in a procedure. This value is then passed back to the main script where the value of **ave** is put into the **result** function. The value in **result** is then used to put the average into the field on the page.

Debugging and Testing

When you begin working with procedures, you need to develop a strategy for keeping track of all your procedures. If you put procedures on invisible buttons on the page on which they are used, you will have great difficulty finding those buttons at a later time. The use of a page or two at the beginning of a folder for Script buttons containing procedures can greatly simplify your debugging.

It is also a good idea to include some documentation on your page of Script buttons. You can build information into the scripts themselves that tell where the procedures are used. Or you might have a field associated with each button that holds a list of the places where you have used that particular procedure.

Project Ideas

Extend the first example given in this chapter so that you can ask the user whether he or she wants to erase the box that is drawn. If the answer is yes, then actually erase the box.

Experiment with the second example given in this chapter. What other mathematical processes might you want to do with procedures? Make a set of math tool procedures.

Create some tool procedures to do useful tasks. Use parameters to pass values to these procedures. Also create some procedures that pass values back. Practice using these procedures at other places in your folder.

Chapter 26. Where Do I Go From Here?

The introduction to this book indicated that it was divided into three general sections. The first few chapters were primarily review and assumed that the reader had experience using *LinkWay* and merely needed to refresh his or her memory. The second part of the book, Chapter 3 through Chapter 20, covered *LinkWay* scripting and the use of a variety of *LinkWay* tools. It introduced computer science ideas, such as programming style and the details of syntax of the commands and functions. The last part of this book focused on programming concepts that are important in most programming languages.

If you have had experience programming in other languages, you no doubt had little difficulty with these last few chapters. In your other programming experience, you encountered conditionals, loops, parameters, and the like. The more programming experience you have had, the easier these last few chapters have seemed.

However, if *LinkWay* scripting is your first programming language, you probably found the last few chapters rather overwhelming. Indeed, many powerful and important ideas are covered rather briefly. Don't be concerned if you feel a bit overwhelmed at this point.

So how do you proceed from here in your work with *LinkWay*? Whether you are a novice or experienced programmer, you should add some other *LinkWay* books to your library and begin to use them. In some books, you will no doubt find other features of *LinkWay* scripting not discussed in this book. Experiment with some of these unfamiliar commands, functions, or tools. Incorporate them into your folders.

You might also consider acquiring some public domain *LinkWay* folders. There are many folders available through catalogs, through users groups, and on networks. Examine folders that interest you. Look at the *LinkWay* code and see how it works. Try modifying some of these folders or adapting some of the ideas you see into your own work.

If you are new to programming, studying examples in other books and examining folders others have written will help you understand the importance of ideas such as the use of conditionals and loops. These other sources will help you see the reasons for local and global variables. They will give you ideas for using parameter passing.

If you are an experienced programmer, looking at reference books and at the work of other programmers will help you see how *LinkWay* scripting is different from other programming languages you know. It will help you understand the ways in which *LinkWay* scripting is different from other programming environments.

Whatever your background or experience, just reading through this book is not enough to master *LinkWay* scripting. Learning to be a good programmer means hours and hours of practice, both at planning your work and at writing code and debugging it.

In addition to the scripting capabilities of *LinkWay*, there are many capabilities for using a variety of media with *LinkWay*. If you are using *LinkWay Live!*, it is even easier to use media such as sound and video. This book has not addressed the use of media at all. You can begin that exploration by looking at the brief overview given in Appendix 6 of this book. You can also read about media tools and media objects in the *LinkWay* manual. After you have studied these sections of your manual, you may want to look for a book that gives you more details and more ideas for using different media. Explore these ideas. The possibilities are endless when you combine the power of your computer using *LinkWay* with the power of a variety of media.

Keep working at it—and have fun!

Appendix 1. Keyboard Shortcuts

Editing Objects in *LinkWay*

Action	Key	Expert mode
Selects next object	Tab	
Delete selected object	D	Lower right target
Edit selected object	E	Lower left target
Move and size selected object	S	Upper right target
Move selected object	M	Upper left target
Change the bounding box color	B	
Exit *LinkWay*	Alt + Q	

In *LinkWay Paint*

Keys	Action
Shift with Copy and Scale	Makes selected graphic transparent
Shift with Box, Bar, Shadow Box	Draws a square box or bar
Shift with Circle	Draw a circle
Shift with Magnify	Picks up color under pointer
Shift with Color selection	Picks up color under pointer
Alt with drawing tools	Forces pointer to character cell boundaries.

In LWEdit

Key	Action
Arrow keys	Moves the cursor one character at a time
Home	Moves the cursor to the beginning of the line
End	Moves the cursor to the end of the line
PgUp	Scrolls text backward one screen
PgDn	Scrolls text forward one screen
Backspace	Deletes one character to the left
Ins	Toggles between insert and replace modes
Del	Deletes the character at the cursor position
F1	Help
F2	Saves a file
F3	Exits LWEdit and saves the text
Shift+F3	Exits LWEdit without saving
F4	Asks for the name of a text file and then copies it into the current text file
Shift + F4	Displays the screen in reference format with the lines the correct length
F5	Searches for a text string
Shift+F5	Inserts the template string %F,color,filename#label% into the text
F6	Reflows a paragraph to the width of the screen
Shift+F6	Reflows a paragraph to the width indicated by the cursor location (minimum = 20 char.)
F7	Copies current line onto "pick" folder
Shift + F7	Inserts the top line from the pick folder
F8	Prints to a specified printer
F9	Deletes the line in which the cursor is located
Shift+F9	Joins the next line to the current one
F10	Inserts a blank line
Shift+F10	Splits a line

Appendix 2. Commands Used in This Book

* indicates that the command is available only in *LinkWay Live!*

bar *color, x1, y1, x2, y2*
beep
bg *colornumber*
*brgb *red, green, blue*
box *color, x1, y1, x2, y2*
*bstyle *number*
call *scriptbuttonname [(value, value…value)]*
case *number*
change *attribute*
clone
cprint
cut *[filename]*
*dcolors *foreground, background*
delete
do *buttonname* do *fieldname* do *listobjectname*
*drowcol *rownumber,columnnumber*
flowset object|*objectname = text-expression*
fmenu
go *pagenumber*
*icolor *number*
if *condition {statement list};* else *{statement list}*
input *prompt1,[prompt2,]variable*
jump *labelname*
line *color, x1, y1, x2, y2*
link *pageid, [foldername]*
match *search string, expression*

mbar
menu
mmode *pointermode*
mouse *[timeout]*
move *left, top, right, bottom*
msg *expression, [expression]*
nombar
noshow
object *objectname*
ocut *[filename]*
odelete
*onshow
opaste *[filename]*
paste *[filename]*
pointer *icon number, xcoordinate, ycoordinate*
print *expression*
proc *[(value, value,…value)]*
prompt *expression*
return *[value]*
set *variable = expression* set object|*objectname = strong*
show
space *number*
stop
tcolors *foreground, background, window*
*text *color,x1,y1,x2,font,text-expression*
var name(length),*[name(length),…name(length)]*
vclear
wait *seconds*
*wmove *left, top, right, bottom*

Appendix 3. Functions Used in This Book

count
chr(*number*)
id
*itime
len
mouseb
mousex
mousey
object
obot
oicon
oleft
oname
oright
otop
otype
owid
*pxwx(*pixelxcoordinate*)
*pywy(*pixelycoordinate*)
result
rnd(*integer*)
seq
time
trunc(*variable*)
vlen (*variable*)
wbot
*wleft
*wright
*wtop

*wxpx(*worldxcoordinate*)
*wypy(*worldycoordinate*)

Appendix 4. Operators

Numerical Operators

+	addition
-	subtraction
*	multiplication
/	division
%	remainder of a division

Numerical Relational Operators

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
=	equal
!=	not equal

Logical Operators

&	and
|	or

String Relational Operators

$<	less than
$<=	less than or equal
$>	greater than
$>=	greater than or equal
$=	equal
$!=	not equal

Rules for Operator Precedence

Operations at a given level are done from the left to the right and only *then* are the operations done on the next level.

1	( )
2	* /
3	+ - :
4	< > = <= >= != $< $> $= $<= $>= $!=
5	& |

Appendix 5. Extensions Used by *LinkWay*

PCC	CGA mode picture file
PCE	EGA mode picture file
PCM	MCGA-256 mode picture file
PCH	MCGA mono mode picture file
PCV	VGA mode picture file
PCZ	VGA-8 mode picture file
PCJ	XGA mode picture file

LWC	CGA folder
LWE	EGA folder
LWH	MCGA folder
LWV	VGA folder
LWM	MCGA-256 folder
LWJ	XGA mode folder
LWZ	VGA-8 mode folder

P10	EGA mode palette
P12	VGA mode palette
P13	MCGA-256 mode palette
P13	XGA mode palette
P13	VGA-8 palette

OB	Cut objects for pasting
PG	Cut pages for pasting
DOC	Document files
REF	Reference files

Appendix 6. Using Media With *LinkWay*

You can use a variety of external devices to work with different media, such as sound and video, in *LinkWay*. If you are using *LinkWay*, you can access these capabilities from the "Useful buttons" choice on the first page of the Main folder. If you are using *LinkWay Live!*, the media capabilities are built in.

The folders available to demonstrate work with media in *LinkWay* include Sound, CDplay, and Video. These folders include information that will help you get started working with media. The Help or Doc buttons in these folders give information on how to use the screens provided.

One of the types of objects that you can create in *LinkWay Live!* is a media object. These media objects can then be used with a variety of multimedia devices. There are several types of media objects you can create.

A Digital Audio Media object can be used to record and play audio data. The captured analog sound is stored digitally on disk. When the sound is played, the file is read and output as an analog signal by the audio playback device. *LinkWay Live!* records and plays audio data in a RIFF format. CD Audio Media objects can be used to play audio data from a compact disk.

A Videodisc Media Object can be used for still or motion sequences. Videodisc objects determine which segments of a videodisc will be played but not where they will be played. Video output can be to a separate video monitor or TV. Audio output can be connected to any audio output device. Video and audio can also be connected to a device that allows the images to be viewed in a window on the screen.

Animated Movie Media Objects can be used to display a series of bitmapped pictures at up to 18 frames per second using special movie players. Creating movies requires special hardware and software, but playing movies does not.

Your *LinkWay Live!* documentation contains a great deal of information on the various forms of media you can work with. It also has detailed material on creating media objects.

Index

{ 184
} 184
| 196
!= 196
$ 197
$!= 197
$< 197
$<= 197
$= 197
$> 197
$>= 197
% 195
& 196
* 195
*/ 115
+ 195
- 195
/ 195
/* 115
< 196
<= 196
= 196
> 196
>= 196
@ 212
Activating buttons
  using do 154
Activating fields
  using do 154
Activating list objects
  using do 155
Addition 195
And 196
Animation 142
  flipping pages 142
  using a loop 144
  using objects 139
  using pictures 139
  using the Page menu 141
  with buttons 139
Answers
  comparing 190
  matching 190
ASCII 109
Autoexec
  buttons 157
Background
  description 8
Background color
  creating 129
  from scripts 128
Backspace key 46
Bakus-Naur notation 79
Bar
  from scripts 125
Base page objects
  editing 21
Base pages 8
  editing 19
  In folder design 100
  scripting with 161
Beep 31
  syntax of 32
Blanks
  in variable values 205
  removing 198, 205
  trailing 205
Bounding box
  finding coordinates of 109
Bounding box targets
  meaning 63
Box
  from scripts 125
Braces
  curley 184
BUG 5
Button
  mouse 149
Button style
  changing 54
Button triggers
  format of 90
Buttons
  activating 24
  as components of folders 9
  copying 24

document 85
icon 21
invisible 22
linking 21
making inactive 53
new 21
picture cut 173
Reference 87
script 29
sign 22
sticky 89
way to represent 21
with procedures 113
Call 114, 218
Captured images
using 122
Capturing pictures 119
Case 190
in scripts 31
Changes
to the screen 165
Character coordinates 37, 108
Characters
deleting 46
inserting 46
Chr 109
Circles
drawing 169
Clone 163
problems with 167
Color
creating 129
from scripts 128, 130
of boxes 130
of icons 130
of objects 130
of text 129
of the background 128
setting mode 12
Color triggers
format of 90
Commands
and functions 76
Comments
symbols for 115
using 115
Concatenation 39
Condition 183
Conditionals 183
Constant
defined 78
Conventions
in folder design 102
Converting
pictures 123
Coordinate systems
character 108
pixel 108
world 108
Coordinates
character 37
converting 111
converting pixel to world 204
converting world to pixel 204
of mouse 111
pixel 41
Copying
button 24
folders 166
pages 163
Copying a line 48
Count 162
Counter
using 212
Cprint 136
Creating graphics
in *LinkWay Paint* 169
Curley braces 184
Cut 165
Cutting
pages 165
Cutting objects 54
D key
to delete objects 62
Dcolor 130
Debug
using 83
Debugging 5
Debugging scripts
using Debug 83
Delete 163
Delete key 46

Deleting a line 47
Deleting characters 46
Deleting objects 54
  in expert mode 63
  the D key 62
Deleting pages 163
Disk space
  saving 122
Display folders 99
Division 195
Do
  command 153
  syntax 153
Document buttons 85
  editing text 87
Documenting scripts 115
Dot commands 135
Double click
  in Format Access Level 24
Drawing
  from scripts 125
  magnify 171
  scaling 170
Drowcol
  with input 176, 180
  with menu 179
E key
  to edit objects 61
Editing
  base page objects 21
  stopping 49
Editing objects
  in expert mode 63
  the E key 61
Editing text
  using LWEdit 92
Else
  with if 185
Enter key 46, 49
Equal 196
Esc key 49
Exit triggers
  format of 90
Expert mode 62
Expression
  defined 78
F10 key 46, 47
F2 key 49
F6 key 49
F7 key 48
  with Shift 48
F7key
  with Shift 48
F9 key 47, 48
Fields
  as components of folders 9
  creating 24
  modes for entering text 25
  size 80
  size of 80
  text in 25
File triggers
  format of 89
Filter folders 99
Flipping pages
  to animate 142
Flowset
  in scripts 109
  syntax 72
  to put text in fields 72
Fmenu 179
  limitations 180
Folder design
  conventions 102
Folders
  designing 95
  display 99
  filter 99
  jump linear 96
  linear 96
  making copy 166
  network 98
  new 13
  point and click 99
  single frame 99
  structure of 95
  tree structure 97
Fonts
  changing 73
  default 73
  in fields 24, 73
  types of in fields 24

Formats
of graphics 123
of pictures 123
Functions
introduction 38
model of 76
Getting started 10
Global 207
Go 30
compared with link 162
syntax 32
Grammar
of *LinkWay* 75
Graphics
converting 123
from other software 121
from scripts 125
in *LinkWay* folders 9
using 122
using part of 122
Graphics formats 123
Graphics mode
setting 12
Greater than 196
Greater than or equal 196
Hardware needed 2
Hypermedia 1
Hypertext 1
creating 89
Icolor 130
Icon 21
Icon color
from scripts 130
Icon numbers 106
Icons
conventions for use 101
in folder design 101
Id
function 162
If 183
using 177
If else 185
If statements
multiple 188
Immediate action scripts 82
Inactive
buttons 53
Input
as operand 32
as parameter 32
checking 190
limitations 176
syntax of 175
to command 32
using 176
Ins key 46
Insert key 46
Insert mode 46
in fields 25
Inserting a line 46
Inserting characters 46
Interactive programming 175
Itime
syntax of 40
Joining lines of text 48
Jump 211
Jump linear folders 96
Key shortcuts 60
Label triggers
format of 90
Labels 143
how created 212
Less than 196
Less than or equal 196
Line 127
Linear folders 96
Link
compared with go 162
problems with 167
Link buttons
size 80
Linking pages
from scripts 162
LinkWay
as an object-like programming environment 10
definition of 7
Starting 10
uses of 7
version of 3

LinkWay Paint
using 15
LinkWay scripting 1
List objects
using 155
Local variables 208
Loop
condition to exit 213
definite 211
test after 213
test before 213
Loop counter 212
Looping 211
template for animation 144
Lower case
in scripts 31
LWCapture
keys to use 120
using 119
LWEdit
keys to use 92
printing from 135
with Document buttons 87
with Reference buttons 88
M key
to move objects 60
Magnifying
drawing 171
Main folder 11
Match 190
Mathematical operators 195
Mbar 166
Memory
releasing 208
Menu
limitations 179
syntax 177
Menu bar
hiding 103, 166
showing 103, 166
Menu command
explanation 178
problems with 178
using 178
mmode 149
Mode
compatibility 12
Modularization
of scripts 215
Modularizing scripts 113
Mouse
low level control 149
syntax 70
to select objects 69
Mouse button
sensing 149
Mouse clicks
how used 81
Mouse coordinates
converting 111
Mouse location
coordinates or 111
Mouseb 149
Mousex 111
Mousey 111
Moving
objects 55
Moving a line 48
Moving and Sizing objects
in expert mode 63
the s key 61
Moving objects
in expert mode 63
the M key 60
Msg
syntax 36
Multiple if statements 188
Multiplication 195
Naming objects
rules 53, 80
Network folders 98
Nmbar 166
Noshow 165
Not equal 196
Numbers
random 186
Object
as a function 71
finding coordinates of 109
properties 105

Object names
in scripts 53, 80
rules 53, 80
Object oriented programming 10
Object target area 23
Object-oriented language 79
Objects
changing order of 81
cutting 54
deleting 54
in *LinkWay* 9, 79
moving 55
order of 81
pasting 54
priority 81
rules for naming 53, 80
selecting 69
Objects on Base page
covering 17
Obot 109
Oicon 106
Oleft 109
Oname 106
Onshow 165
Operand
as input 32
Operations
order of 198
Operator precedence 198
Operators
mathematical 195
string 197
Or 196
Order
changing object 81
of objects 81
Order of operations 198
Oright 109
Otop 109
Otype 107, 148
Page
base 8
Pages
cutting 165
deleting 163
linking from scripts 162
making copy 163
new 14
number of 162
pasting 165
restoring 164
scripting with 161
Paint text 16
Parameter
called input 32
Parameters 205
scope of 217
using 217
Paste 165
Pasting
pages 165
Pasting objects 54
Picture cut buttons 173
Picture formats 123
Pictures
as components of folders 9
blank 17
capturing 119
converting 123
printing 134
using 122
using part of 122
Pixel coordinates 41, 108
converting to world 204
Point and click folder 99
Pointers 148, 205
Prerequisites 2
Print 136
Print page
from Page menu 133
Print Screen 120
Printing 133
changing size of images 133
controlling style 135
folder 133
from LWEdit 135
from scripts 136
page 133
pictures 134
text files 135
using LWEdit 135
using Tools 133

Proc 114, 218
Procedure buttons
  placing 115
  using 115
Procedures
  on buttons 113
  placing 115
  returning values 220
  running 115
  writing 114
Profile
  in *LinkWay Live!* 62
Prompt 31, 35
  syntax 33
Properties
  of objects 105
PrSc 120
Pxwx 111, 204
Pywy 111, 204
Quoted string 33
Random numbers 186
Reference buttons
  creating 87
  editing 88
  size 80
  with triggers 89
Reflowing text 49
Regular action scripts 82
Rejoining lines of text 48
Remainder 195
Remarks
  symbols for 115
  using 115
Removing blanks 205
Removing windows 12
Replace mode 46
  in fields 25
Retrace 104
Return 114, 220
Return triggers
  format of 90
Returning values 220
Rnd 186
S key
  to move and size objects 61
Scaling
  drawings 170
Scope of variables 221
  problems with 209
Screen updates
  controlling 165
Script buttons 29
  size 80
Scripting
  defined 81
Scripts
  documenting 115
  immediate action 82
  modularizing 113
  regular action 82
  to create graphics 125
  use of case in 31
Selecting objects
  from scripts 52
  Tab key 59
  with the mouse 69
Seq 143
Set
  complete syntax 77
  full syntax 201
  syntax 71
  to put text in fields 70
Shift F7 48
Shift key
  in *LinkWay Paint* 169
Show 165
Sign
  of a button 22
Single frame folders 99
Size
  of text in fields 80
  of text on buttons 80
Space 190
Speed
  improving 122
Splitting a line 47
Squares
  drawing 169
Starting
  automatic 157

Statement
defined 82
Status line
hiding 103
showing 103
Sticky button 89
Stop 213
Stopping editing 49
String operators 197
Strings
comparison 197
concatenating 39
putting together 39
Style
of printing 135
Subscripts
in variables 206
Subtraction 195
Syntax 32
Syntax diagrams
explanation of 78
Tab key
to select objects 59
Target area
object 23
Tcolors 129
Text
in fields 25
paint 16, 25
paint versus text in fields 25
Text color
from scripts 129, 130
Text editor
using 45
Text files
printing 135
Text in fields
from scripts 70
Text in *LinkWay Paint*
using 16
Text Pop-up buttons
size 80
Time
syntax of 39
Tool buttons
getting 65
Tools
DragObj 65
in *LinkWay* 63
Story 65
Tracker 64
Tracker button
getting 65
Tracker tool 64
Trailing blanks 205
removing 198, 205
Tree structure
of folders 97
Triggers
format of button 90
format of color 90
format of exit 90
format of file 89
format of label 90
format of return 90
introduction 89
Trunc 198, 205
Type
in fields 24
Upper case
in scripts 31
Values
returning 220
Values of variables
blanks in 205
Var 201
Variables 201
clearing 208
global 207
length of 205
local 208
parameters 205
pointer 205
releasing memory 208
restrictions on names of 201
scope of 207, 221
selecting a part of 206
size of 201
subscripts in 206
summary 217
Vclear 208
Version of *LinkWay* 3

Vlen 205
Wait 31
  decimal inputs 144
  syntax 32
Wbot 110
Windows
  removing 12
Wleft 110
Word wrap
  in fields 25
World coordinates 108
  converting to pixel 204
Wright 110
Wtop 110
Wxpx 204
Wypy 204

Put the Future of Education in Your Hands...



Pick up David Thornburg's visionary books that explore the social and technological trends shaping education.

Edutrends 2010—Restructuring, Technology, and the Future of Education

Thornburg examines the recent flurry of teacher bashing, moving from the reasoning flaws of those who would dismantle U.S. public education to the change process necessary for educational restructuring for the 21st century.

He proposes a way to think intelligently about the future while we reshape education to meet the needs of our society.

Education, Technology, and Paradigms of Change for the 21st Century

Learn unique patterns of thinking that encourage students to go above and beyond the three Rs. The technologies of calculators, computers, laser discs, and video games converge with new concepts of learning and intelligence to provide unprecedented opportunities for all students. Let this progressive book lead you toward a future filled with new paradigms.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

Make This Step Your Next Step.

You've got your new Mac... Now what?

You want to know what those cute icons mean. You need to find out how to use that mouse-thing-a-ma-jig.

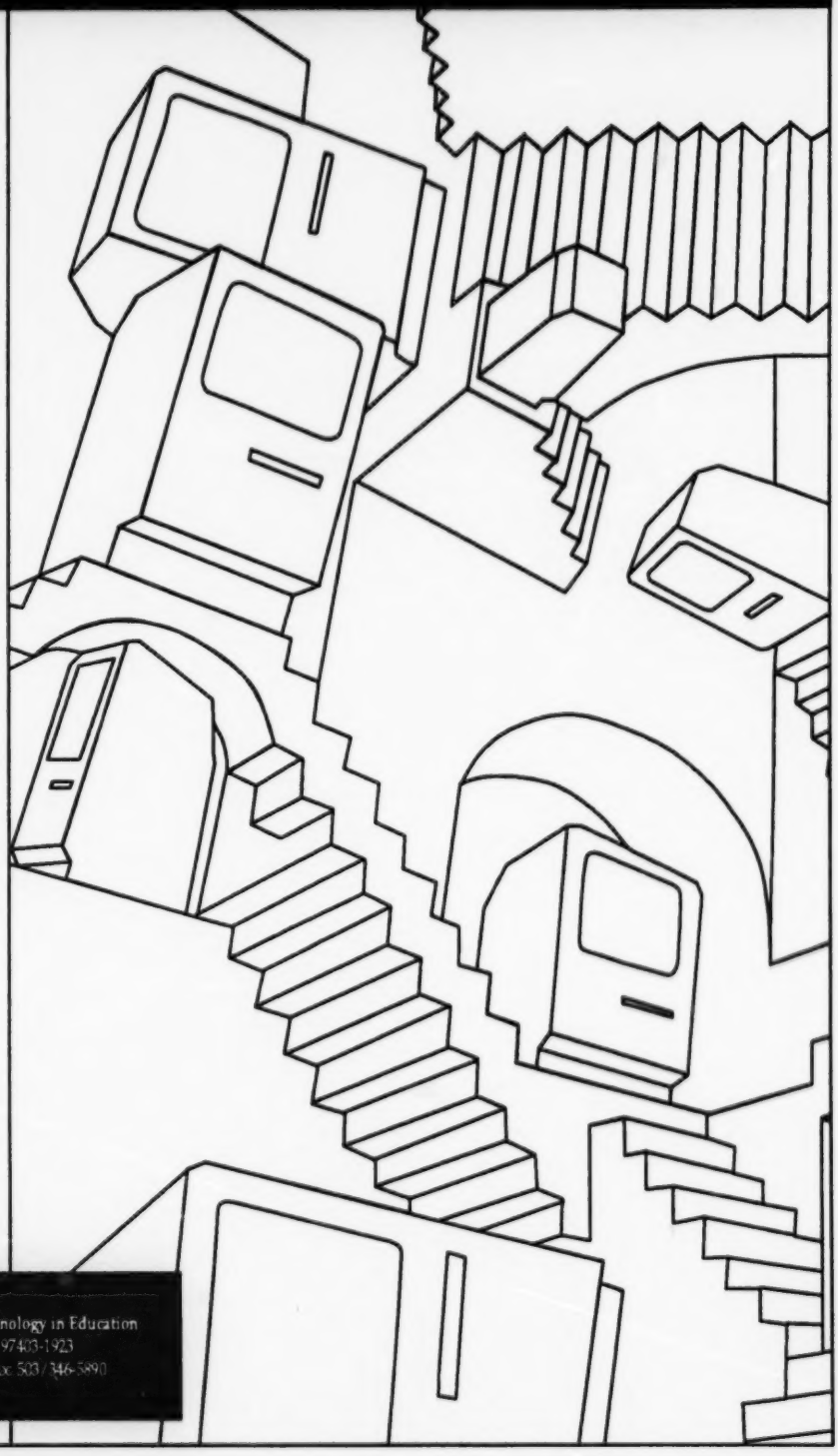
You'd like to know how to "empty the trash," and it'd be great to understand exactly what the "chooser" chooses.

By following the footsteps in ISTE's new book, *Macintosh Step by Step*, you'll pick up on all these basic Macintosh operating procedures, plus more—from finding the "power" button to demystifying the little Apple menu in the upper left corner.

And it also teaches you the theory behind Mac file storage and the importance of backing up your data on a regular basis.

In all, you'll find that this simple, illustrated guide will lead you on your way toward becoming a Mac user—not just a Mac owner.

Just call today and make your next computing step a step in the ISTE direction.



ISTE

International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

All you need is the right tool...

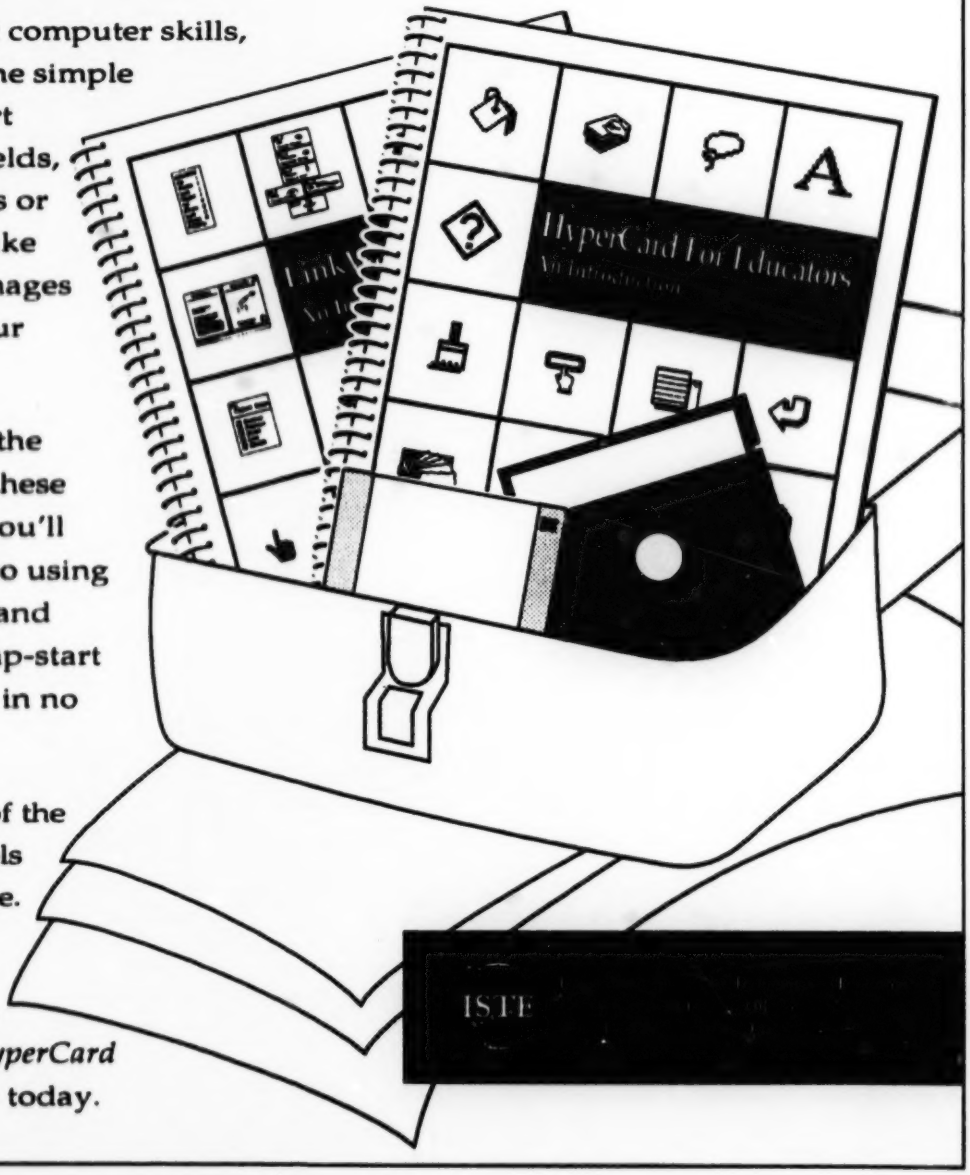
With either *HyperCard for Educators* (Macintosh) or *LinkWay for Educators* (MS-DOS) as an introduction to the interactive world of hypermedia, you can learn to build libraries of knowledge suitable for both classroom and personal use.

If you have basic computer skills, you can follow the simple tutorials and start using buttons, fields, and backgrounds or base pages to make your data and images come alive in your classroom.

By starting with the foundation that these books provide, you'll be on your way to using sounds, images, and animation to jump-start your curriculum in no time!

Take advantage of the organizational tools computers provide.

Call ISTE to order and begin building your *HyperCard* or *LinkWay* skills today.



ISTE Books & Courseware Order Form

Name _____

School/Business _____

Address _____

City _____ State _____ Zip/Postal Code _____

Country _____ Phone _____

☐ Yes! I'm presently an ISTE member. Membership number must be included to receive discount. My membership number is _____ (Discount does not apply to initial membership fee.)

Occupation

Grade Level: ☐ all (K-12) Specific grade level _____

- ☐ Teacher (350) Subject _____
- ☐ Administrator (320) Title/Position _____
- ☐ Higher Education (360) Title/Position _____
- ☐ Federal/State Government (310) Title/Position _____
- ☐ Educational Information Resource Manager (IRM) (320)
- ☐ Technology Coordinator (340)
- ☐ Other _____

Terms & Conditions

Terms: Net 60 orders must be accompanied by authorized company purchase orders. Orders for individuals must be pre-paid, C.O.D., or charged to a Visa, MasterCard, or Discover card number. All non-U.S. orders must be pre-paid.
Shipping: Shipping method is UPS. Add 5% in addition to shipping rate for first class delivery. ***Allow three weeks for delivery.*** Call for rush orders.
Prices: Subject to change without notice.
Returns: ISTE allows return of merchandise, provided the materials are in resellable condition. There will be a 10% restocking charge. All returns must be accompanied by an ISTE Return Authorization form. If you wish to return merchandise, please contact the Distribution Dept. *GST Registration #128828431*

Shipping & Handling for Books & Courseware

$0-$15.99 (subtotal) ..... add $3.25
$16-$45.99 (subtotal) $4.50
$46-$75.99 (subtotal) $5.50
$76-$100.99 (subtotal) $6.50
$101 or more ..... 7% of subtotal

Do not include *additional site license fees or subscription costs* when computing shipping rates.

Non-U.S. orders for Books & Courseware are sent surface mail. If you want your Books & Courseware order shipped AIR-MAIL, please check here. ☐ ISTE will bill you the additional shipping charge.

Payment Options

☐ **Payment enclosed.** (Make checks out to ISTE.) Non-U.S. orders must be pre-paid with U.S. funds or credit card.
☐ VISA ☐ MasterCard ☐ Discover Card Exp. Date _____

☐ **Purchase Order enclosed.** (Please add $2.50 for order processing. P.O. not including $2.50 fee will be returned.)

☐ **C.O.D.** for U.S. Book orders only. You will pay UPS the total upon delivery (check or cash—ISTE will add $2.75 order processing).

GST Registration Number 128828431

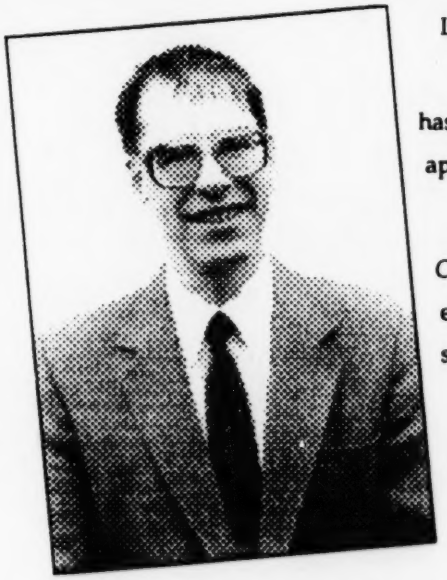
ISTE Books, Courseware, & Nonmember Subscriptions

Qnty.	Title	Member Unit Price	Nonmember Unit Price	Total Price

BOOKS & COURSEWARE SUBTOTAL	
Add Shipping, based on Books & Courseware SUBTOTAL	+
Add $10 S & H for Don't Copy that Floppy	+
Add Additional 5% of SUBTOTAL if shipped to PO Box, AK, HI, or outside U.S.	+
Add 7% of SUBTOTAL for GST if shipped to Canada	+
If billed with purchase order, add $2.50; If COD, add $2.75	+
	=
Membership Total from other side	+
TOTAL	=

MAIL this form to:
ISTE, 1787 Agate St., Eugene, OR 97403-1923 USA
FAX this form with credit card or P.O. information to: 503/346-5890
PHONE your credit card or P.O. order to:
ISTE Order Desk: 800/336-5191

Dave Moursund knows...



ISTE Executive Officer Dave Moursund is dedicated to improving education.

Since he first began working with computers in education in 1963, Dave has constantly sought to improve education at all levels through the use of appropriate computer-related technology.

As a continuing part of this effort, Dave's newest book—*The Technology Coordinator*—discusses the goals of education and how teachers and teachers of teachers can join in the enormous change process facing today's schools.

This book was written for educators who want to play a leadership role in the instructional use of technology. Dave uses interviews with technology coordinators to illustrate the needs, roles, and qualifications needed to accomplish the goals of educational technology.

In recent years, Dave has focused much of his writing efforts on problem solving and on staff development. Some of Dave's other books also available from ISTE include *Problem-Solving Models for Computer Literacy, LogoWriter for Educators: A Problem Solving Approach, Logo PLUS for Educators: A Problem Solving Approach* (co-authored with Sharon Yoder), and *Effective Inservice for Integrating Computer-as-Tool into the Curriculum.*

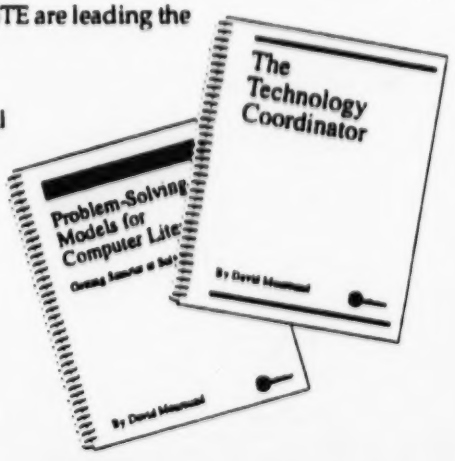
"Our educational system is not nearly as good as it could be," says Dave. "We know that computer-related technology can make substantial contributions to improving both the content of the curriculum and the instructional delivery system. Such technology can empower students and teachers."

ISTE believes that computer-related technology must become a tool that students and teachers use routinely if students are to be adequately prepared for adult citizenship in our Information Age society.

ISTE's wide range of periodicals, publications, projects, and other activities mirrors this vision and commitment.

Call us today to order Dave's latest books, and discover how he and ISTE are leading the way for the educators and school systems of tomorrow.

ISTE. Proudly putting you in touch with the world of educational technology.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Order Desk: 800/336-5191 Fax: 503/346-5890

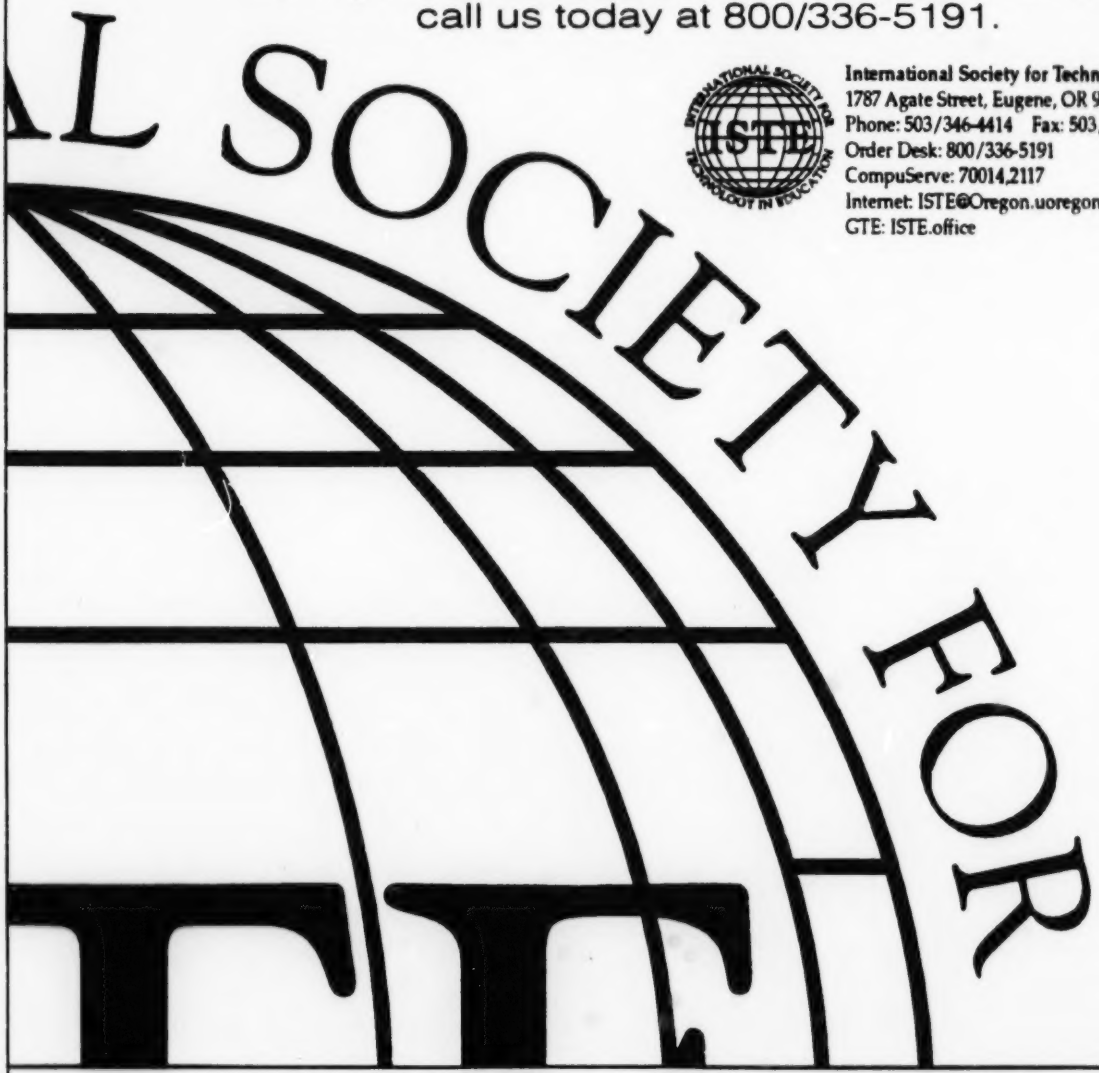
ISTE BRINGS THE WORLD OF TECHNOLOGY CLOSER TO YOU.

By drawing from the resources of committed professionals worldwide, ISTE provides support that helps educators like yourself prepare for the future of education.

ISTE members benefit from the wide variety of publications, specialized courseware, and professional organizations available to them.

They also can enjoy exciting conferences, global peer networking, and mind-expanding independent study courses.

So if you're interested in the education of tomorrow, call us today at 800/336-5191.



International Society for Technology in Education
1787 Agate Street, Eugene, OR 97403-1923
Phone: 503/346-4414 Fax: 503/346-5890
Order Desk: 800/336-5191
CompuServe: 70014,2117
Internet: ISTE@Oregon.uoregon.edu
GTE: ISTE.office

WE'LL PUT YOU IN TOUCH WITH THE WORLD.